CSN08x14

Scripting for Cybersecurity and Networks

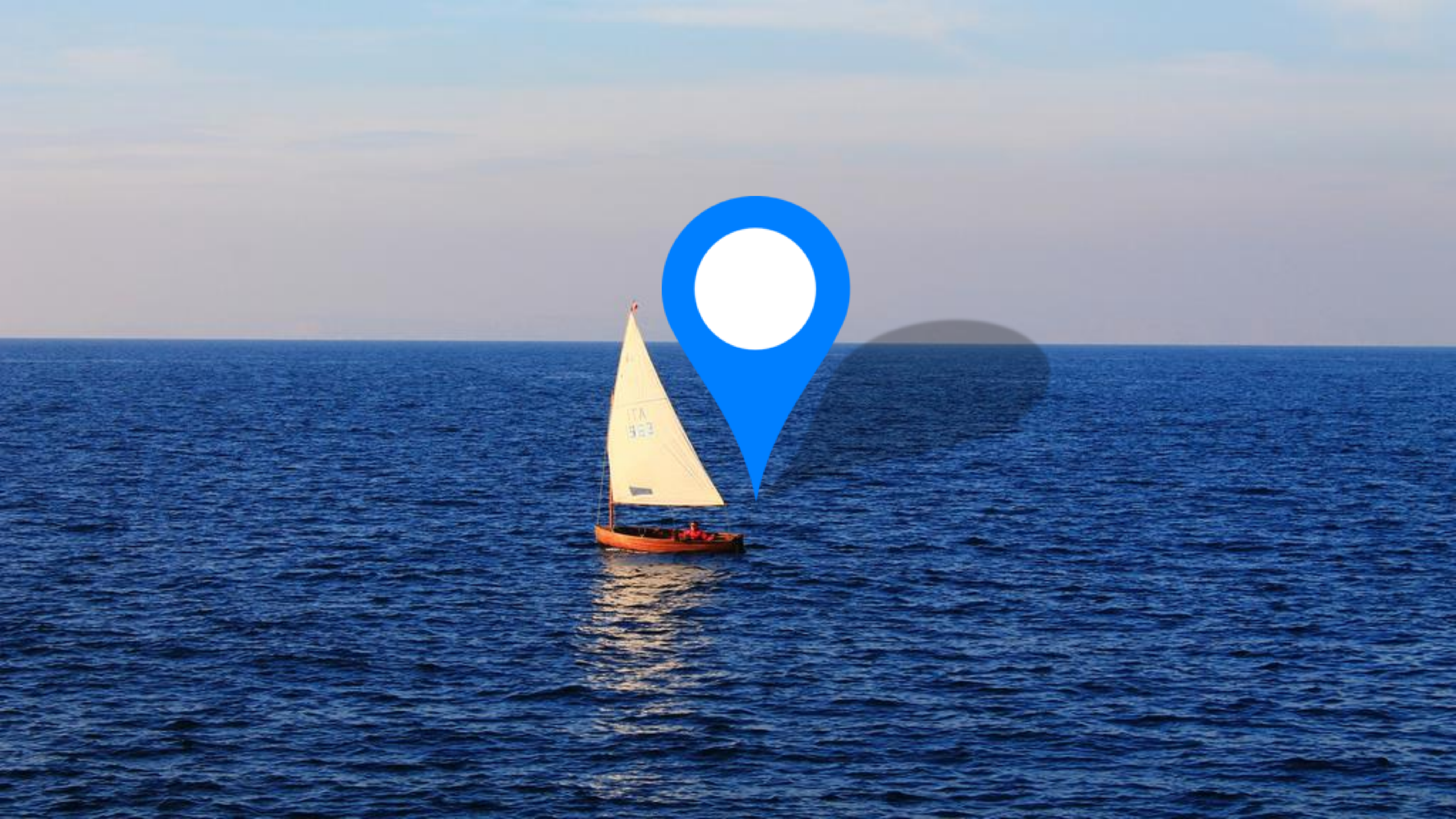
Lecture 11:

Geolocation with Python

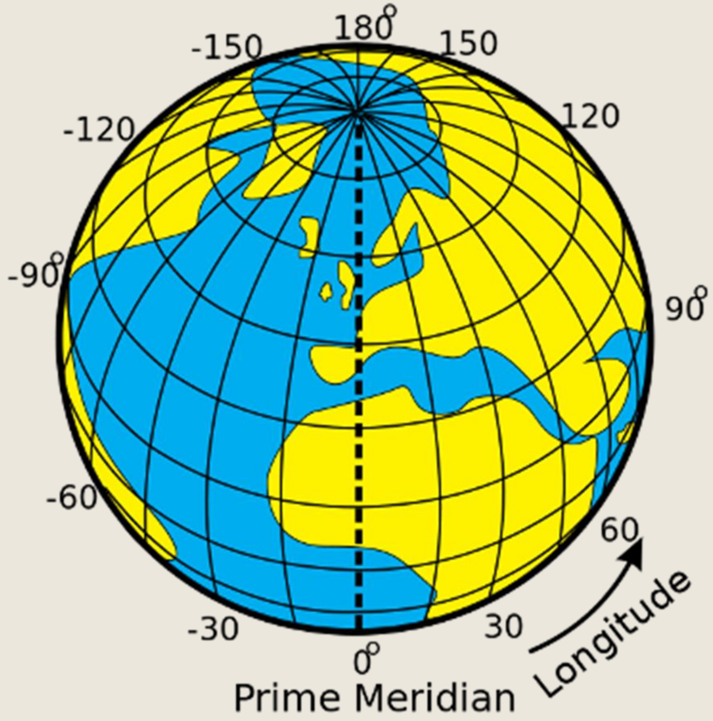
In this lecture

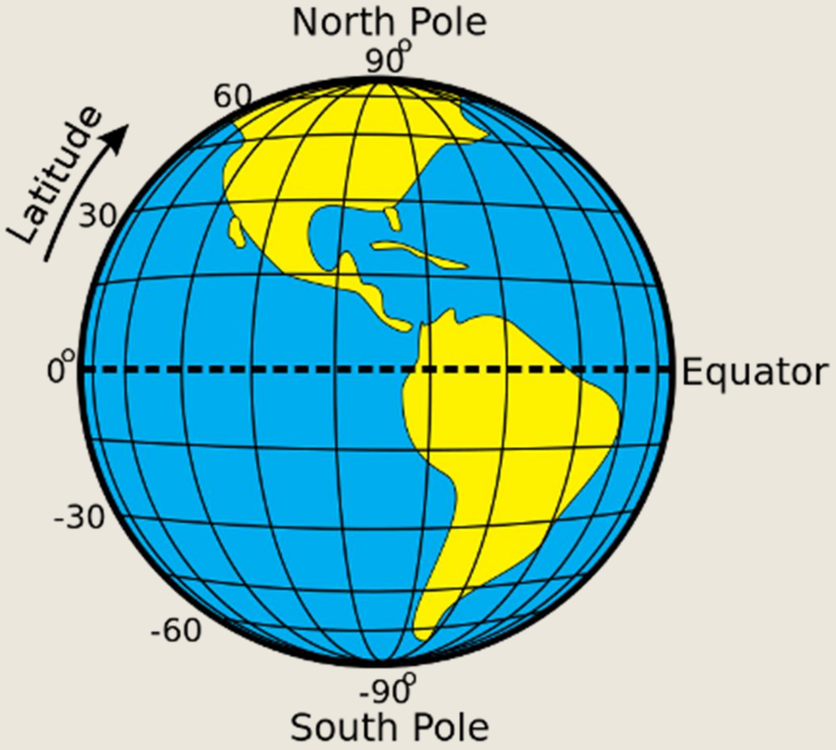
- Concepts
  - *Geolocation and mapping*
  - *KML – google earth / google maps*
  - *EXIF information in digital images*
- Python modules:
  - *geoip2 - geo-location module (we use the sub-module geoip2.database)*
  - *simplekml - KML module*
  - *PIL, PIL.ExifTags – modules for extracting EXIF information from images (these are installed via pip install pillow)*

Geolocation and mapping


ITA
583

Geolocation





- **Longitude:**
  0-180° West, 0-180° East
  0 ° "prime" meridian
  placed arbitrarily (agreed by international treaty)

- **Latitude:**
  0-90° North, 0-90° South
  0° = Equator
  90° North = North Pole, 90° South = South Pole

Pag. 120.
60. 120. 200. 260. 320.
Miles of Great Britain
A New Map of
EUROPE
According to yᵉ Neweſt Obſervations
By
H. Moll Geographer.
The first Meridian from Ferro Iſland
THE WESTERN OCEAN
NORTHERN OCEAN
Artick Circle
German Ocean
North Cape
N. Zembla
Lapland
SWEDEN
NORWAY
DENMARK
Baltick Sea
Botnick B.
IRELAND
GREAT BRITAIN
English ch.
FRANCE
GERMANY
POLAND
SPAIN
PORTUGAL
ITALY
Gulf of Venice
TURKY in EUROPE
MUSCOVY
EUXINE or BLACK
Constantinople
Natolia
Levant
Candia
Cyprus
Sicily
Sardinia
Corsica
MEDITERRANEAN SEA
BARBARY
PART OF AFRICA
Straits of Gibraltar
B. of Biscay

(Longitude, Latitude, (Altitude)) uniquely identify every place on earth - Look up on map

https://commons.wikimedia.org/wiki/File:Herman_Moll_A_New_Map_of_Europe_According_to_the_Newest_Observations_1721.JPG
http://www.rmg.co.uk/discover/explore/prime-meridian-Greenwich
The Greenwich Meridian was chosen as the Prime Meridian of the World in 1884. Delegates from 25 nations met in Washington DC for the International Meridian Conference.
Greenwich won the prize of Longitude 0º by 22 to 1 against (San Domingo), with 2 abstentions (France and Brazil).

Geolocation – search google maps etc

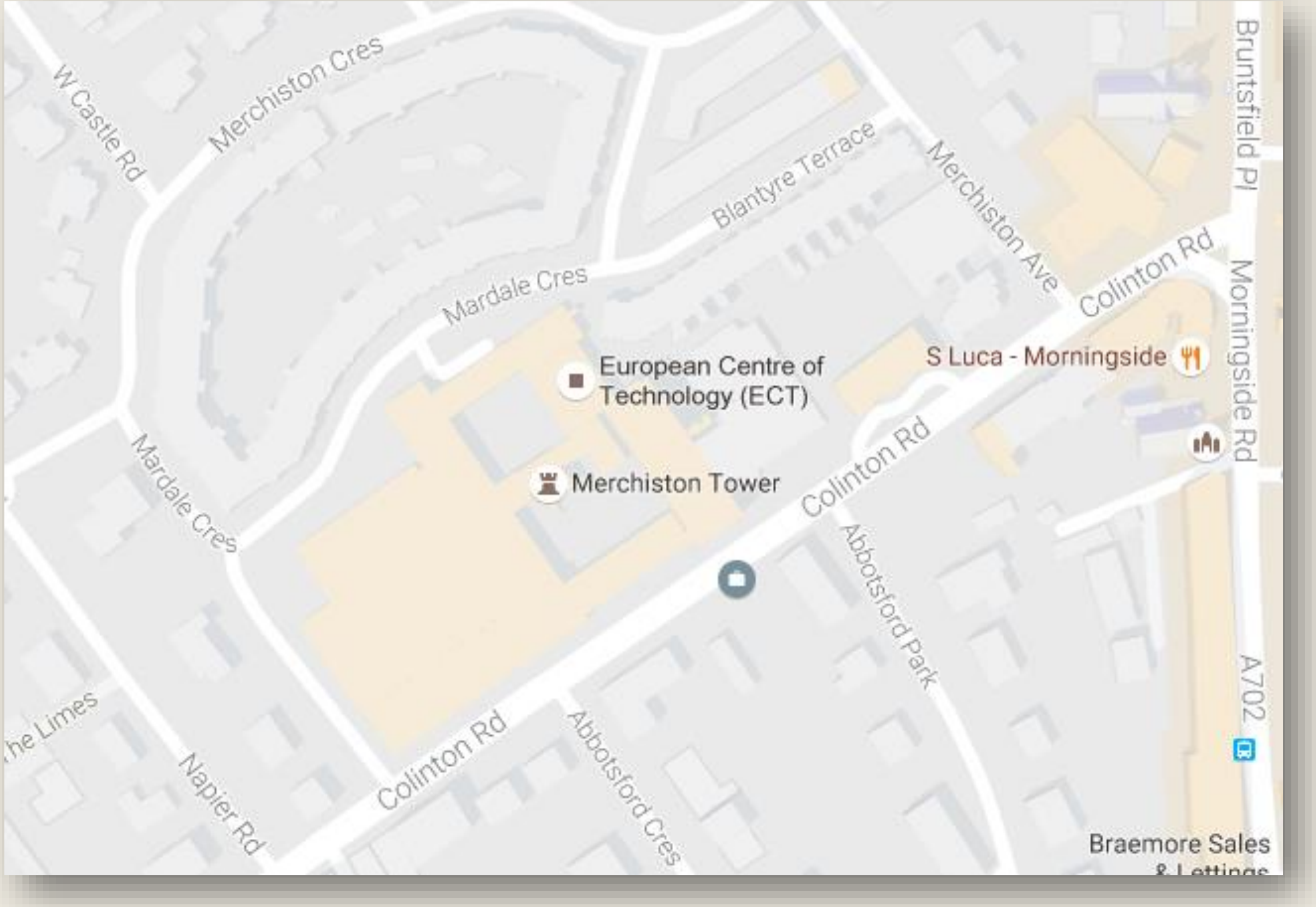
Traditional:
3° 12' 49" W, 55° 55' 58" N

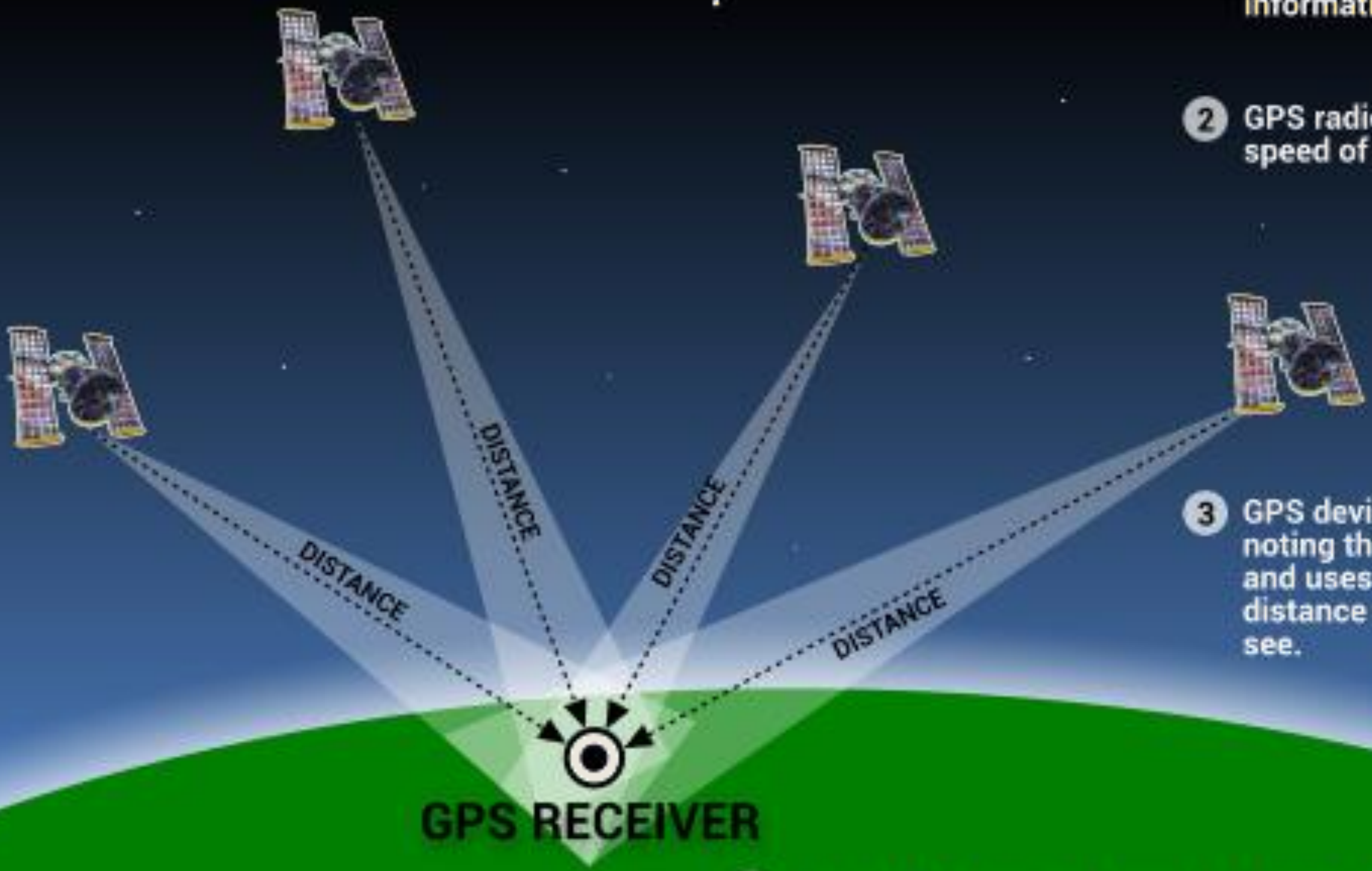
Decimal:
(-3.2136 , 55.932892)

Negative longitude: West

Negative latitude: South

Longitude ALWAYS comes first for mapping apps!!!





1. Each satellite broadcast radio signals with their location, statuses and precise time information.

2. GPS radio signal travels at speed of light ~ 300,000 km/h.

3. GPS device receives radio signals, noting their exact time of arrival and uses these to calculate its distance from each satellite it can see.

4. Once a GPS receiver knows its distance from at least 4 satellites, it uses geometry to determine its exact location on Earth in 3D.

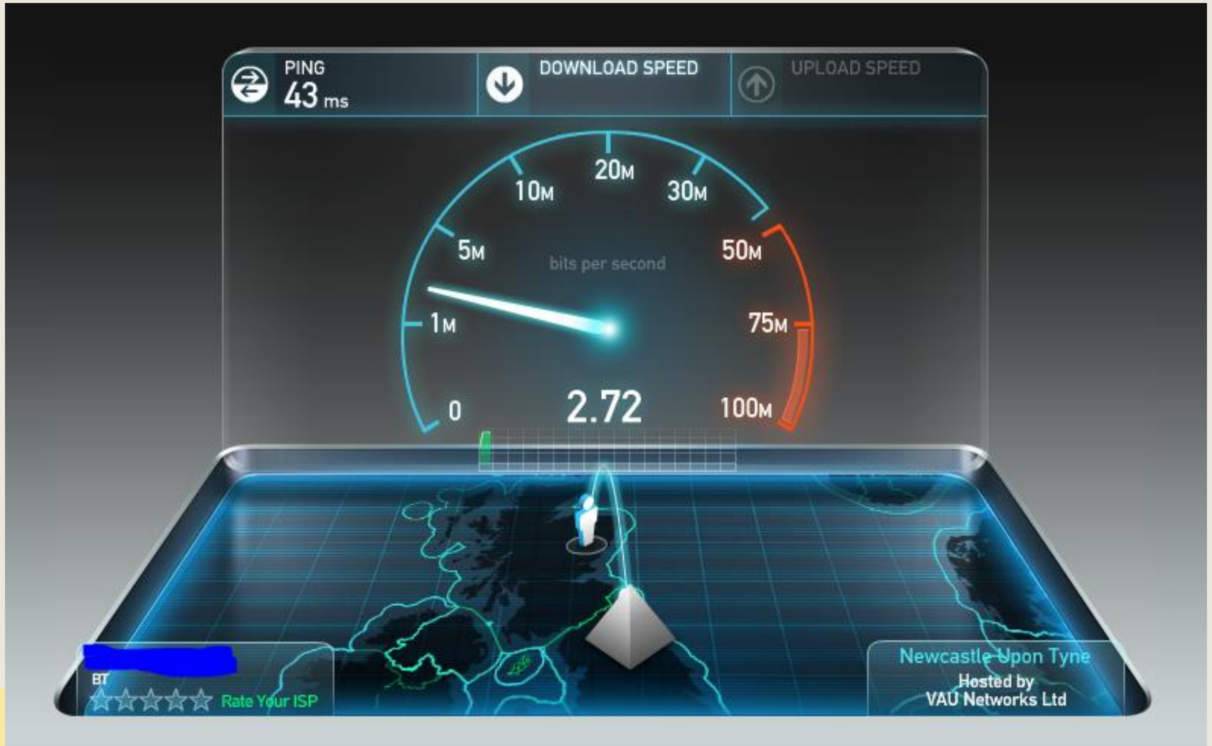
Uses of geolocation in computing

Uses of geolocation in general Computing

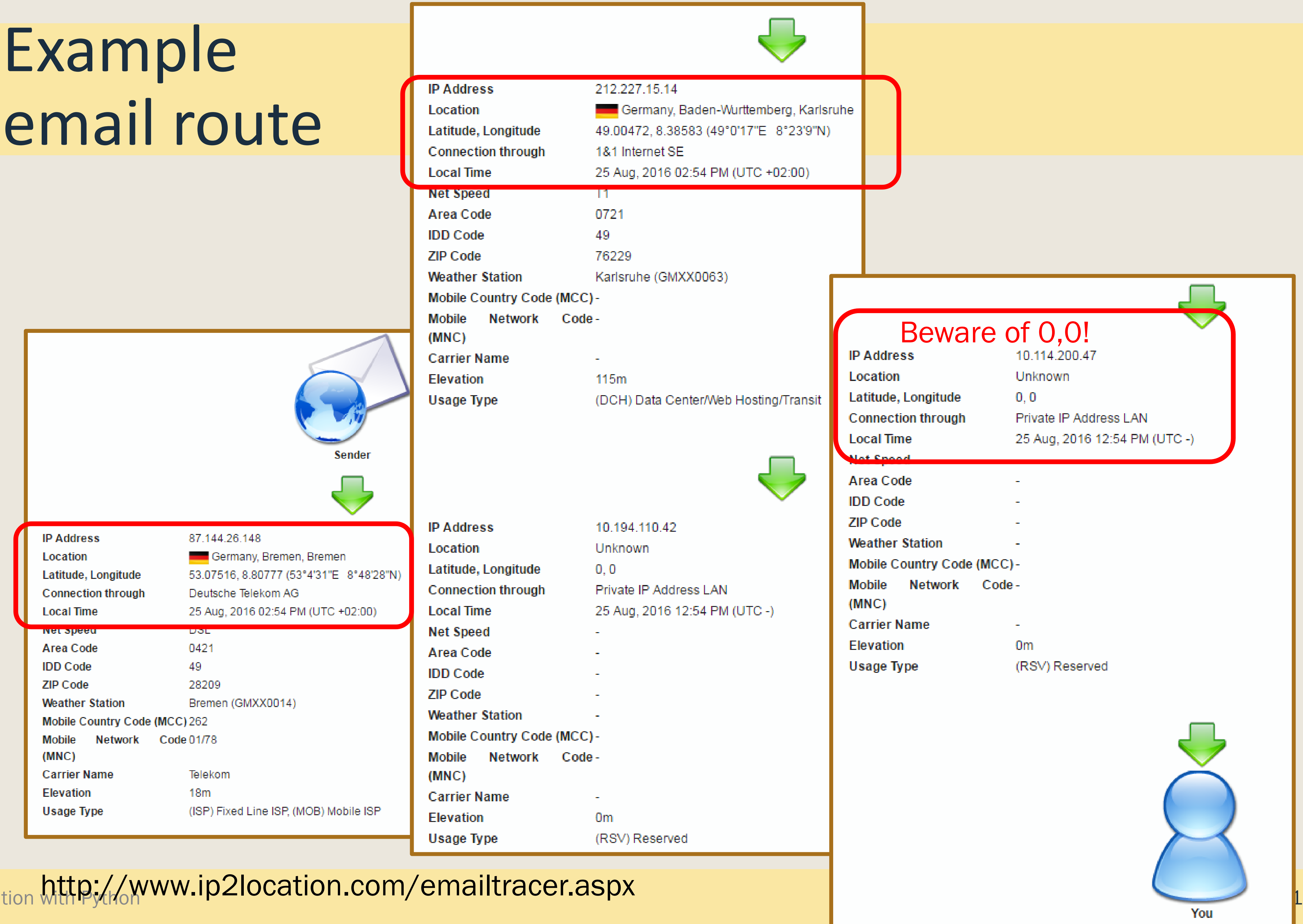
- Satnav!
- Web design: deliver appropriate content / language to user
- Mobile phones:
  - *Map*
  - *Nearby – takeaways, bus stops, cinemas, …*
- Social media:
  - *Twitter*
  - *Facebook*
  - *Instagram etc*

Uses of geolocation in Networking and Computer Security

- Mapping of IP addresses:
  - *Origins of an attack*
  - *Origins/destinations of network traffic*
  - *Route taken by an email*
  - *Measure of network speed via ping distance*
- Block traffic from / to certain countries



Example email route



IP Address	87.144.26.148
Location	Germany, Bremen, Bremen
Latitude, Longitude	53.07516, 8.80777 (53°4'31"E 8°48'28"N)
Connection through	Deutsche Telekom AG
Local Time	25 Aug, 2016 02:54 PM (UTC +02:00)
Net Speed	DSL
Area Code	0421
IDD Code	49
ZIP Code	28209
Weather Station	Bremen (GMXX0014)
Mobile Country Code (MCC)	262
Mobile Network Code (MNC)	01/78
Carrier Name	Telekom
Elevation	18m
Usage Type	(ISP) Fixed Line ISP, (MOB) Mobile ISP

IP Address	212.227.15.14
Location	Germany, Baden-Wurttemberg, Karlsruhe
Latitude, Longitude	49.00472, 8.38583 (49°0'17"E 8°23'9"N)
Connection through	1&1 Internet SE
Local Time	25 Aug, 2016 02:54 PM (UTC +02:00)
Net Speed	T1
Area Code	0721
IDD Code	49
ZIP Code	76229
Weather Station	Karlsruhe (GMXX0063)
Mobile Country Code (MCC)	-
Mobile Network Code (MNC)	-
Carrier Name	-
Elevation	115m
Usage Type	(DCH) Data Center/Web Hosting/Transit

IP Address	10.194.110.42
Location	Unknown
Latitude, Longitude	0, 0
Connection through	Private IP Address LAN
Local Time	25 Aug, 2016 12:54 PM (UTC -)
Net Speed	-
Area Code	-
IDD Code	-
ZIP Code	-
Weather Station	-
Mobile Country Code (MCC)	-
Mobile Network Code (MNC)	-
Carrier Name	-
Elevation	0m
Usage Type	(RSV) Reserved

Beware of 0,0!

IP Address	10.114.200.47
Location	Unknown
Latitude, Longitude	0, 0
Connection through	Private IP Address LAN
Local Time	25 Aug, 2016 12:54 PM (UTC -)
Net Speed	
Area Code	-
IDD Code	-
ZIP Code	-
Weather Station	-
Mobile Country Code (MCC)	-
Mobile Network Code (MNC)	-
Carrier Name	-
Elevation	0m
Usage Type	(RSV) Reserved

http://www.ip2location.com/emailtracer.aspx

Uses of geolocation in digital forensics

- Laptops / mobile devices connecting to wifi – where and when?
- Mobile phones store location
  - *Where were phone calls made /received?*
  - *Where was a suspect / victim when they visited a certain website?*
  - *Evidence from maps / other apps*
- Satnav – e.g. location saved every 10 seconds
- Photos taken by mobile phone / camera with GPS/wifi:
  - *EXIF photo metadata stores location*

SatNav example

- This example shows GPS info recorded every 10 seconds
- Data stored in an SQLite database

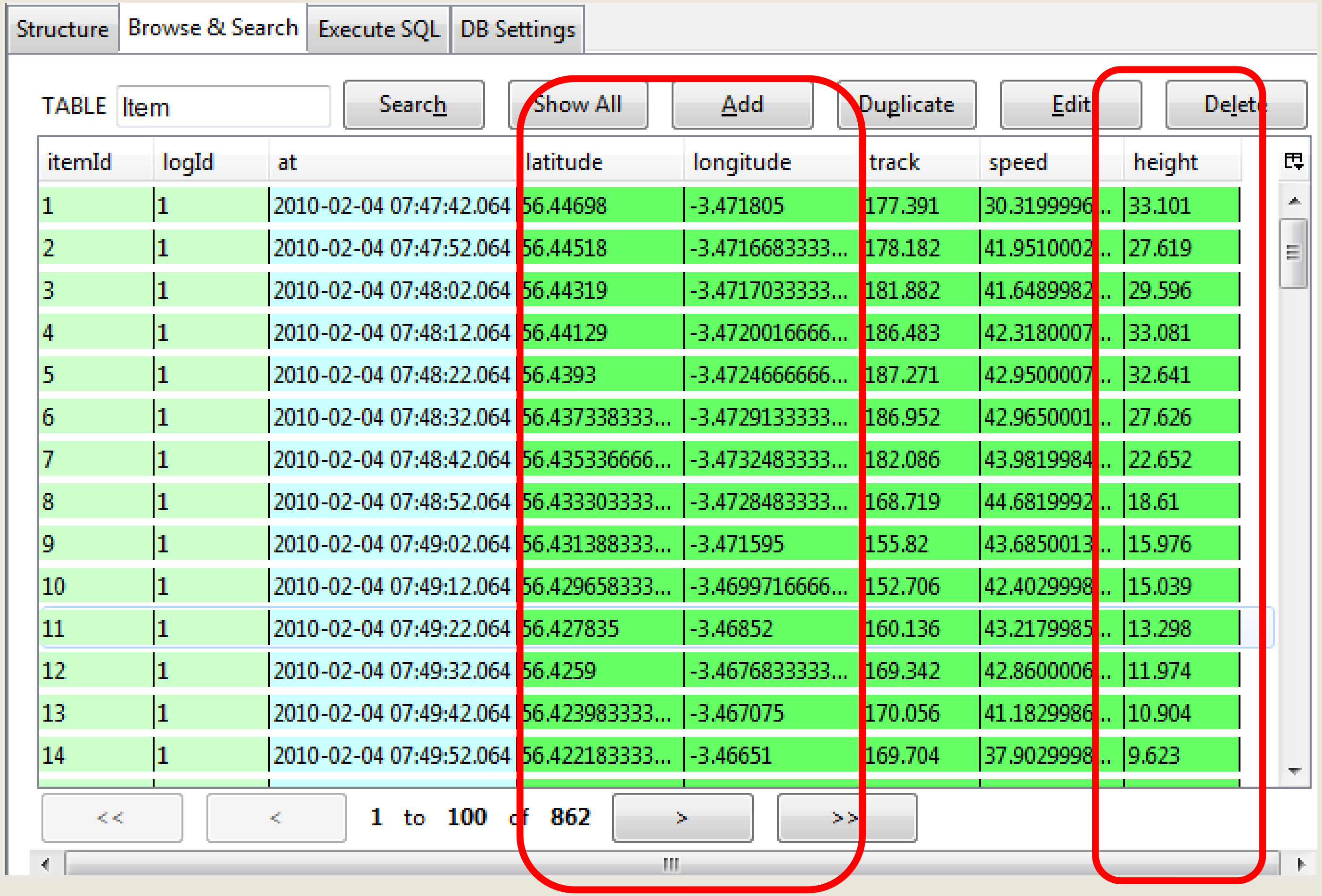


TABLE Item

itemId	logId	at	latitude	longitude	track	speed	height
1	1	2010-02-04 07:47:42.064	56.44698	-3.471805	177.391	30.3199996..	33.101
2	1	2010-02-04 07:47:52.064	56.44518	-3.4716683333...	178.182	41.9510002..	27.619
3	1	2010-02-04 07:48:02.064	56.44319	-3.4717033333...	181.882	41.6489982..	29.596
4	1	2010-02-04 07:48:12.064	56.44129	-3.4720016666...	186.483	42.3180007..	33.081
5	1	2010-02-04 07:48:22.064	56.4393	-3.4724666666...	187.271	42.9500007..	32.641
6	1	2010-02-04 07:48:32.064	56.437338333...	-3.4729133333...	186.952	42.9650001..	27.626
7	1	2010-02-04 07:48:42.064	56.435336666...	-3.4732483333...	182.086	43.9819984..	22.652
8	1	2010-02-04 07:48:52.064	56.433303333...	-3.4728483333...	168.719	44.6819992..	18.61
9	1	2010-02-04 07:49:02.064	56.431388333...	-3.471595	155.82	43.6850013..	15.976
10	1	2010-02-04 07:49:12.064	56.429658333...	-3.4699716666...	152.706	42.4029998..	15.039
11	1	2010-02-04 07:49:22.064	56.427835	-3.46852	160.136	43.2179985..	13.298
12	1	2010-02-04 07:49:32.064	56.4259	-3.4676833333...	169.342	42.8600006..	11.974
13	1	2010-02-04 07:49:42.064	56.423983333...	-3.467075	170.056	41.1829986..	10.904
14	1	2010-02-04 07:49:52.064	56.422183333...	-3.46651	169.704	37.9029998..	9.623

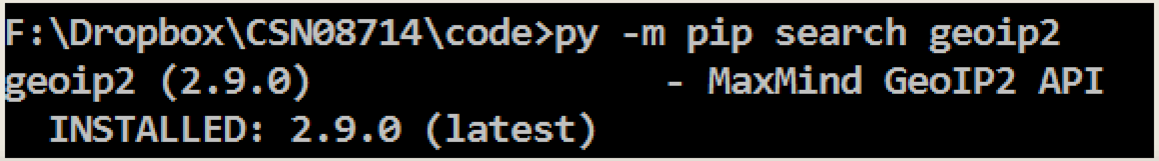
1 to 100 of 862

geoip2.database: Geolocation for IP Addresses

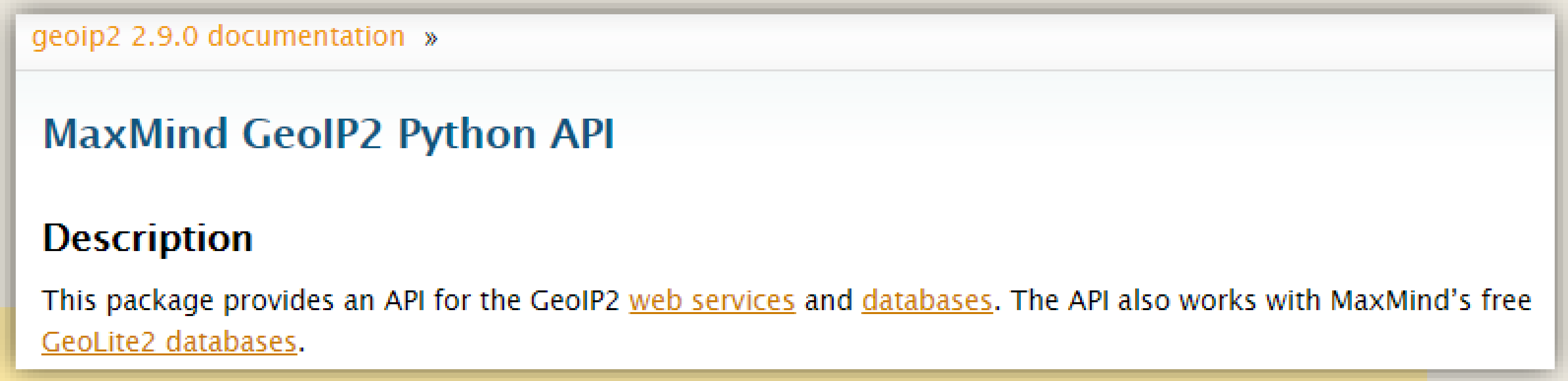
geoip2 module

- API to look up geolocation and related information for a given IP address
  - *A little bit like "whois" for Python, but focused entirely on geolocation information*
- Requires a suitable database, e.g. GeoIP2 (some are available free)
- We specifically use the sub-module **geoip2.database**

See https://geoip2.readthedocs.io/en/latest/, https://dev.maxmind.com/geoip/geoip2/whats-new-in-geoip2/



```
F:\Dropbox\CSN08714\code>py -m pip search geoip2
geoip2 (2.9.0)               - MaxMind GeoIP2 API
  INSTALLED: 2.9.0 (latest)
```

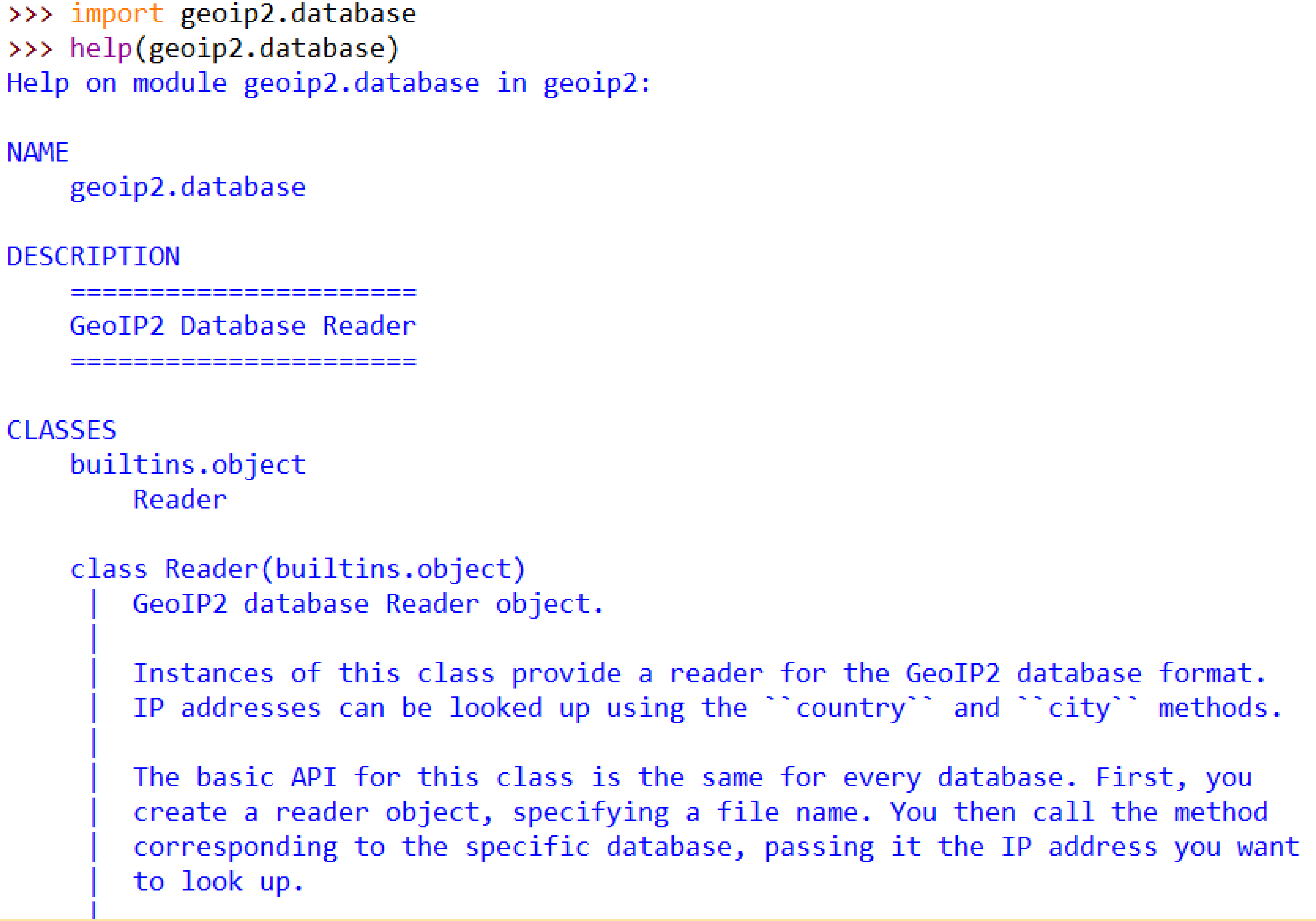


geoip2 2.9.0 documentation »

MaxMind GeoIP2 Python API

Description

This package provides an API for the GeoIP2 web services and databases. The API also works with MaxMind's free GeoLite2 databases.



```
>>> import geoip2.database
>>> help(geoip2.database)
Help on module geoip2.database in geoip2:

NAME
    geoip2.database

DESCRIPTION
    ======================
    GeoIP2 Database Reader
    ======================

CLASSES
    builtins.object
        Reader

    class Reader(builtins.object)
     |  GeoIP2 database Reader object.
     |
     |  Instances of this class provide a reader for the GeoIP2 database format.
     |  IP addresses can be looked up using the ``country`` and ``city`` methods.
     |
     |  The basic API for this class is the same for every database. First, you
     |  create a reader object, specifying a file name. You then call the method
     |  corresponding to the specific database, passing it the IP address you want
     |  to look up.
     |
```

Preparation

- Install **geoip2** module with pip

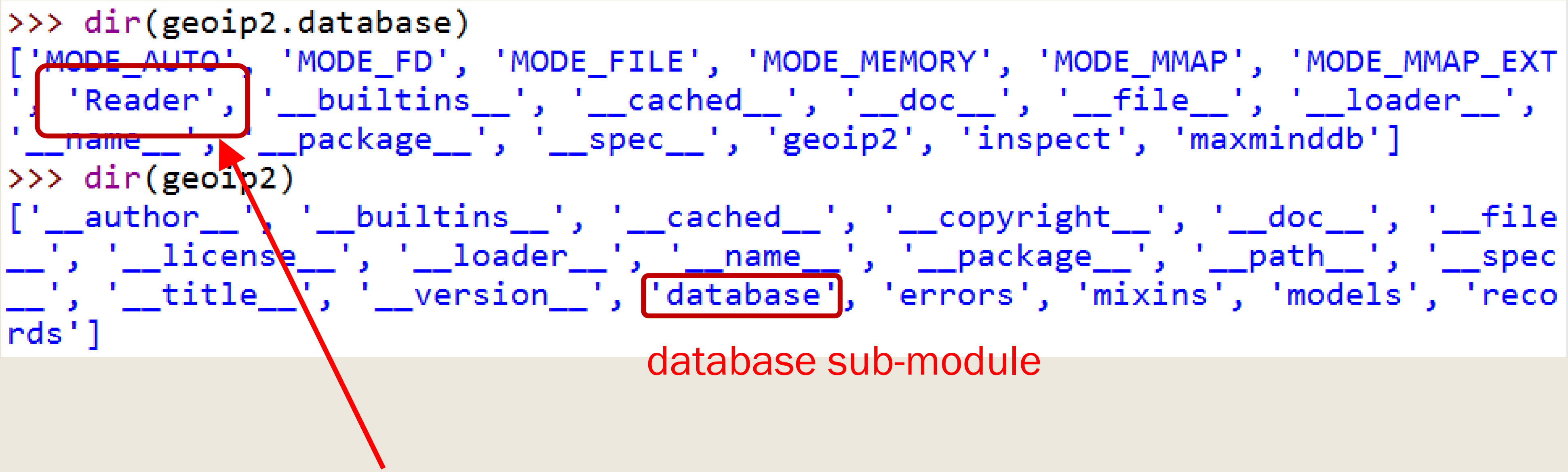
  > python -m pip install geoip2 (from windows command line)

- Download a suitable geolocation **database** and put it in the Python37 directory
  - *e.g. download open-source GeoLite2 City database from MaxMind available at https://dev.maxmind.com/geoip/geoip2/geolite2/*
  - *Unzip it*
  - *Copy/move it to Python37 directory and note the name and path (for me, Geo.mmdb)*

```
Geo.mmdb                    13/11/2018 19:52     MMDB File          52,549 KB
```

Using the geoip2 module: first steps

- Import geoip2.database
- Create a Reader object
  - *This should be closed when you're finished*



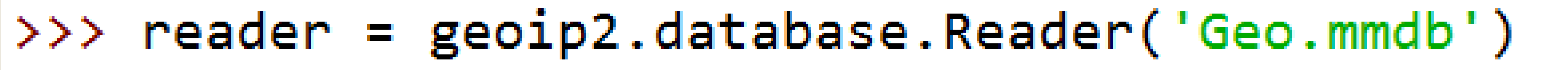
```
>>> dir(geoip2.database)
['MODE_AUTO', 'MODE_FD', 'MODE_FILE', 'MODE_MEMORY', 'MODE_MMAP', 'MODE_MMAP_EXT
', 'Reader', '__builtins__', '__cached__', '__doc__', '__file__', '__loader__',
'__name__', '__package__', '__spec__', 'geoip2', 'inspect', 'maxminddb']
>>> dir(geoip2)
['__author__', '__builtins__', '__cached__', '__copyright__', '__doc__', '__file
__', '__license__', '__loader__', '__name__', '__package__', '__path__', '__spec
__', '__title__', '__version__', 'database', 'errors', 'mixins', 'models', 'reco
rds']
```

database sub-module

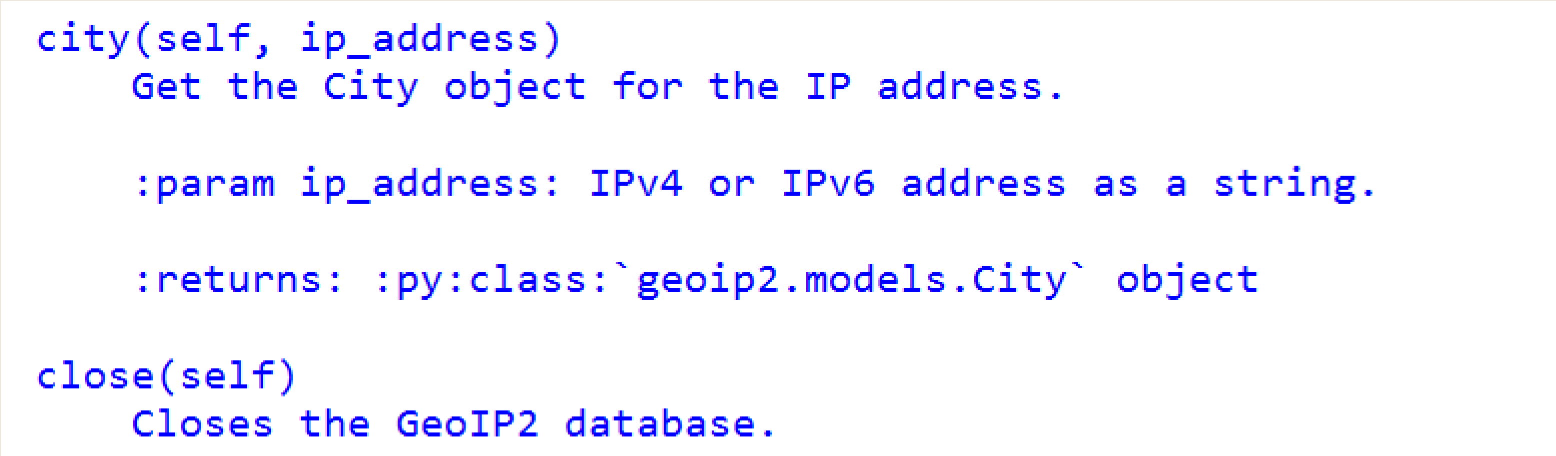
function used to create a Reader object - effectively this parses the geolocation database. Information can then be extracted from this for a given IP address

Using the geoip2 module: Reader object methods

- Methods for Reader objects

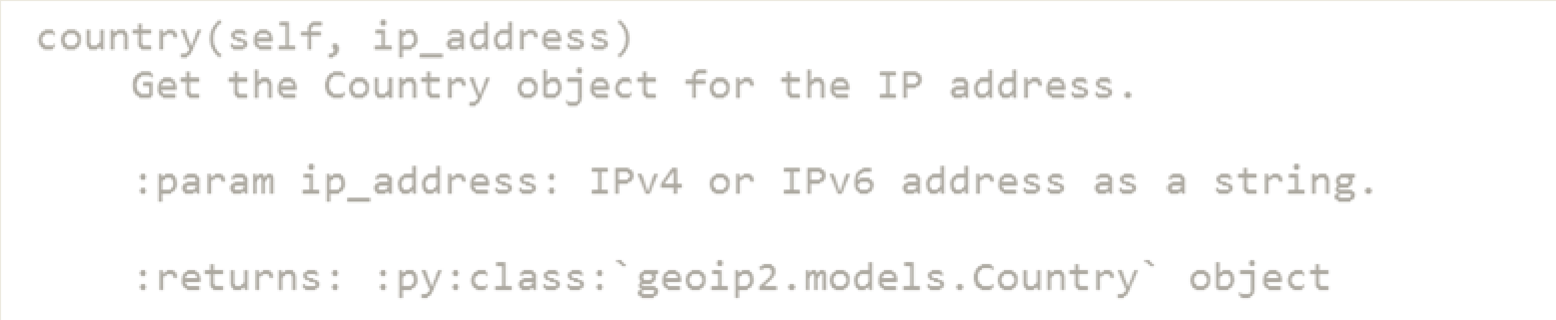


```
>>> reader = geoip2.database.Reader('Geo.mmdb')
```



```
city(self, ip_address)
    Get the City object for the IP address.

    :param ip_address: IPv4 or IPv6 address as a string.

    :returns: :py:class:`geoip2.models.City` object

close(self)
    Closes the GeoIP2 database.
```

These are the two main methods for us which work with the GeoLite2 City database

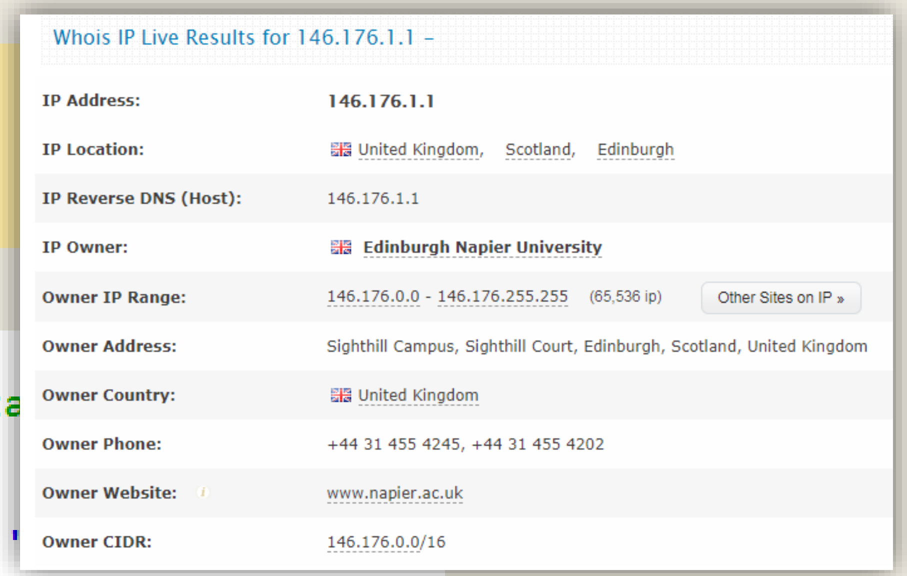


```
country(self, ip_address)
    Get the Country object for the IP address.

    :param ip_address: IPv4 or IPv6 address as a string.

    :returns: :py:class:`geoip2.models.Country` object
```

To use the country method, you would need the country database

city method example

```
>>> import geoip2.database
>>> reader = geoip2.database.Reader(r'C:\Users\40009856\AppData
ython\Python37\Geo.mmdb')
>>> print(reader.city('146.176.1.1'))
geoip2.models.City({'city': {'geoname_id': 2650225, 'names': {'
 'en': 'Edinburgh', 'es': 'Edimburgo', 'fr': 'Édimbourg', 'ja': 'エディンバラ', '
pt-BR': 'Edimburgo', 'ru': 'Эдинбург', 'zh-CN': '爱丁堡'}}, 'continent': {'code'
: 'EU', 'geoname_id': 6255148, 'names': {'de': 'Europa', 'en': 'Europe', 'es': '
Europa', 'fr': 'Europe', 'ja': 'ヨーロッパ', 'pt-BR': 'Europa', 'ru': 'Европа', '
zh-CN': '欧洲'}}, 'country': {'geoname_id': 2635167, 'is_in_european_union': Tru
e, 'iso_code': 'GB', 'names': {'de': 'Vereinigtes Königreich', 'en': 'United Kin
gdom', 'es': 'Reino Unido', 'fr': 'Royaume-Uni', 'ja': 'イギリス', 'pt-BR': 'Rein
o Unido', 'ru': 'Великобритания', 'zh-CN': '英国'}}, 'location': {'accuracy_radi
us': 5, 'latitude': 55.9521, 'longitude': -3.1965, 'time_zone': 'Europe/London'}
, 'postal': {'code': 'EH1'}, 'registered_country': {'geoname_id': 2635167, 'is_i
n_european_union': True, 'iso_code': 'GB', 'names': {'de': 'Vereinigtes Königrei
ch', 'en': 'United Kingdom', 'es': 'Reino Unido', 'fr': 'Royaume-Uni', 'ja': 'イ
ギリス', 'pt-BR': 'Reino Unido', 'ru': 'Великобритания', 'zh-CN': '英国'}}, 'subd
ivisions': [{'geoname_id': 2638360, 'iso_code': 'SCT', 'names': {'de': 'Schottla
nd', 'en': 'Scotland', 'es': 'Escocia', 'fr': 'Ecosse', 'pt-BR': 'Escócia', 'ru'
: 'Шотландия', 'zh-CN': '苏格兰'}}, {'geoname_id': 3333229, 'iso_code': 'EDH', '
names': {'en': 'Edinburgh'}}], 'traits': {'ip_address': '146.176.1.1'}}, ['en'])
>>>
```



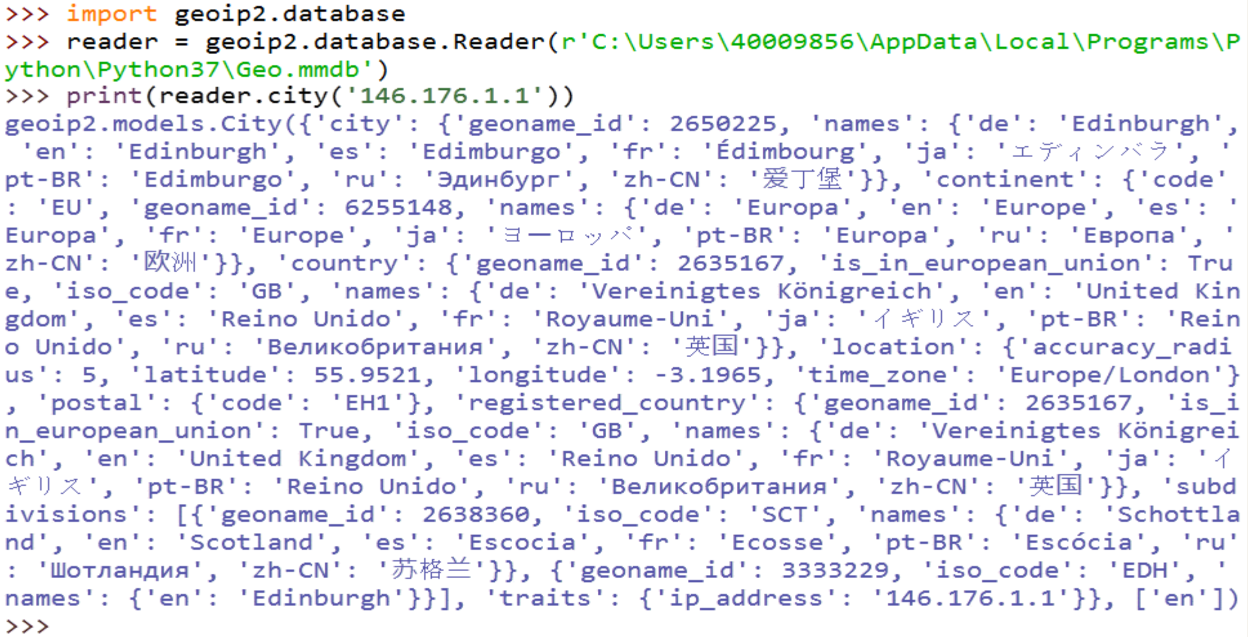
Whois IP Live Results for 146.176.1.1 –

IP Address:	146.176.1.1
IP Location:	United Kingdom, Scotland, Edinburgh
IP Reverse DNS (Host):	146.176.1.1
IP Owner:	Edinburgh Napier University
Owner IP Range:	146.176.0.0 - 146.176.255.255 (65,536 ip) Other Sites on IP »
Owner Address:	Sighthill Campus, Sighthill Court, Edinburgh, Scotland, United Kingdom
Owner Country:	United Kingdom
Owner Phone:	+44 31 455 4245, +44 31 455 4202
Owner Website:	www.napier.ac.uk
Owner CIDR:	146.176.0.0/16

Scripting for Cybersec & Networks

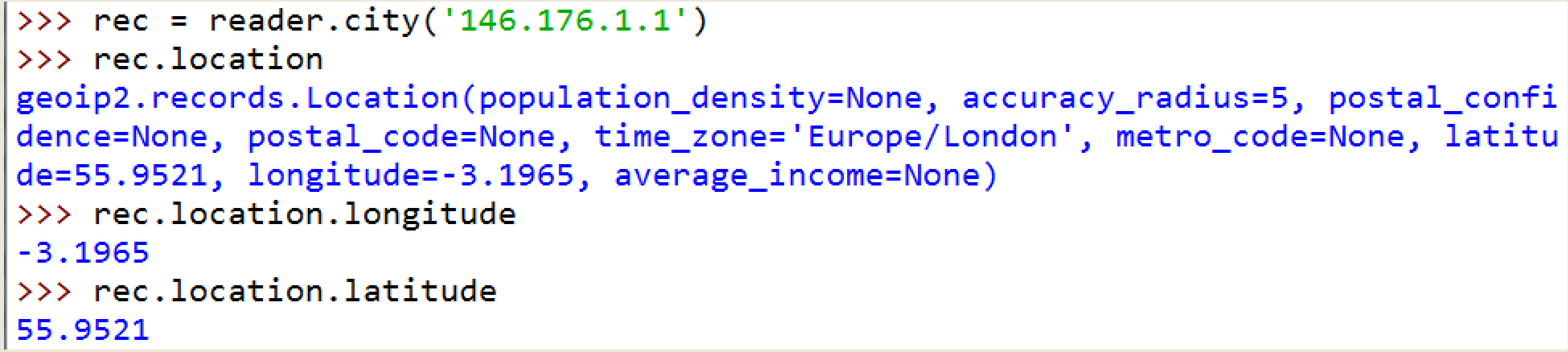
city method example continued

- Rather than this single lookup, which gives quite an unwieldy result, we can get the attributes we want through more targeted methods



```
>>> import geoip2.database
>>> reader = geoip2.database.Reader(r'C:\Users\40009856\AppData\Local\Programs\P
ython\Python37\Geo.mmdb')
>>> print(reader.city('146.176.1.1'))
geoip2.models.City({'city': {'geoname_id': 2650225, 'names': {'de': 'Edinburgh',
 'en': 'Edinburgh', 'es': 'Edimburgo', 'fr': 'Édimbourg', 'ja': 'エディンバラ', '
pt-BR': 'Edimburgo', 'ru': 'Эдинбург', 'zh-CN': '爱丁堡'}}, 'continent': {'code'
: 'EU', 'geoname_id': 6255148, 'names': {'de': 'Europa', 'en': 'Europe', 'es': '
Europa', 'fr': 'Europe', 'ja': 'ヨーロッパ', 'pt-BR': 'Europa', 'ru': 'Европа', '
zh-CN': '欧洲'}}, 'country': {'geoname_id': 2635167, 'is_in_european_union': Tru
e, 'iso_code': 'GB', 'names': {'de': 'Vereinigtes Königreich', 'en': 'United Kin
gdom', 'es': 'Reino Unido', 'fr': 'Royaume-Uni', 'ja': 'イギリス', 'pt-BR': 'Rein
o Unido', 'ru': 'Великобритания', 'zh-CN': '英国'}}, 'location': {'accuracy_radi
us': 5, 'latitude': 55.9521, 'longitude': -3.1965, 'time_zone': 'Europe/London'}
, 'postal': {'code': 'EH1'}, 'registered_country': {'geoname_id': 2635167, 'is_i
n_european_union': True, 'iso_code': 'GB', 'names': {'de': 'Vereinigtes Königrei
ch', 'en': 'United Kingdom', 'es': 'Reino Unido', 'fr': 'Royaume-Uni', 'ja': 'イ
ギリス', 'pt-BR': 'Reino Unido', 'ru': 'Великобритания', 'zh-CN': '英国'}}, 'subd
ivisions': [{'geoname_id': 2638360, 'iso_code': 'SCT', 'names': {'de': 'Schottla
nd', 'en': 'Scotland', 'es': 'Escocia', 'fr': 'Ecosse', 'pt-BR': 'Escócia', 'ru'
: 'Шотландия', 'zh-CN': '苏格兰'}}, {'geoname_id': 3333229, 'iso_code': 'EDH', '
names': {'en': 'Edinburgh'}}], 'traits': {'ip_address': '146.176.1.1'}}, ['en'])
>>>
```

- Examples:



```
>>> rec = reader.city('146.176.1.1')
>>> rec.location
geoip2.records.Location(population_density=None, accuracy_radius=5, postal_confi
dence=None, postal_code=None, time_zone='Europe/London', metro_code=None, latitu
de=55.9521, longitude=-3.1965, average_income=None)
>>> rec.location.longitude
-3.1965
>>> rec.location.latitude
55.9521
```

KML:
Keyhole Markup Language

Mapping a location in google maps / earth etc

- To map a location in google maps / earth and similar tools, we need:
  - *The location: Longitude & Latitude (& Altitude)*
- Can be mapped directly, just paste into search bar
  - *But what if we want to map 100 locations, possibly with other info such as timestamps, route, etc?*

→ Write info to a KML file

→ *use API to interact (beyond scope of this module)*

→ *Open KML file in Google maps / Google earth*

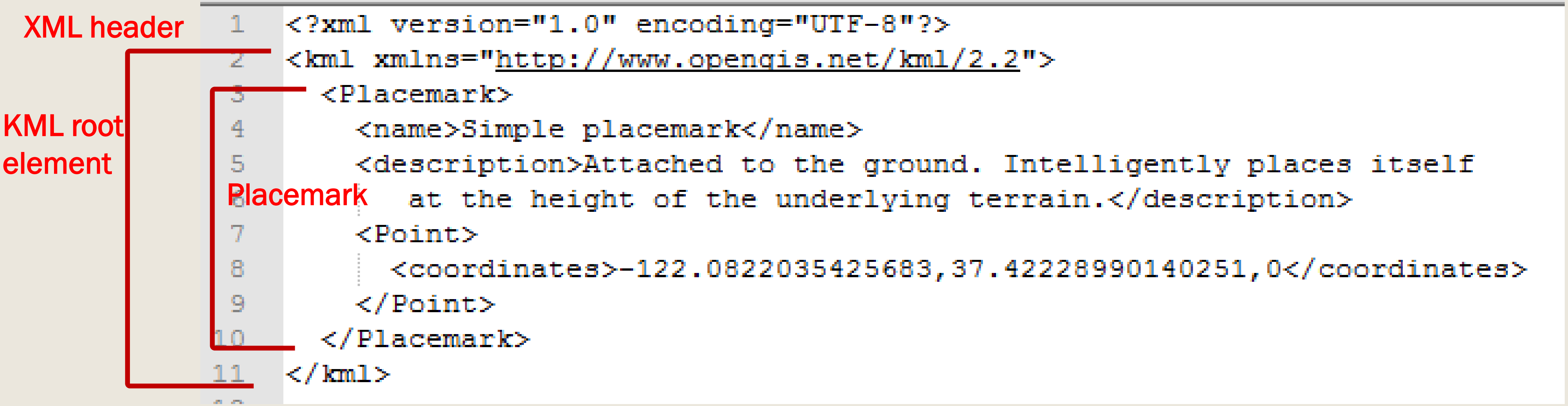
What is KML?

- KML: Keyhole Markup Language: Part of the XML "family"
- used for geospatial data
- Developed for use with Google Earth*
  - *Now widely used by geospatial software*
- KML file format: .kml extension / .kmz extension (compressed)
- KML files can pinpoint locations, add image overlays, and expose rich data in new ways.
- KML is an international standard maintained by the Open Geospatial Consortium, Inc. (OGC).

* Google Earth was originally developed by Keyhole, Inc and called Keyhole Earth Viewer. Bought by Google in 2004

KML

- Must have: one XML header, one KML root element
- Typical location element: Placemark



```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
  <Placemark>
    <name>Simple placemark</name>
    <description>Attached to the ground. Intelligently places itself
       at the height of the underlying terrain.</description>
    <Point>
      <coordinates>-122.0822035425683,37.42228990140251,0</coordinates>
    </Point>
  </Placemark>
</kml>
```

XML header

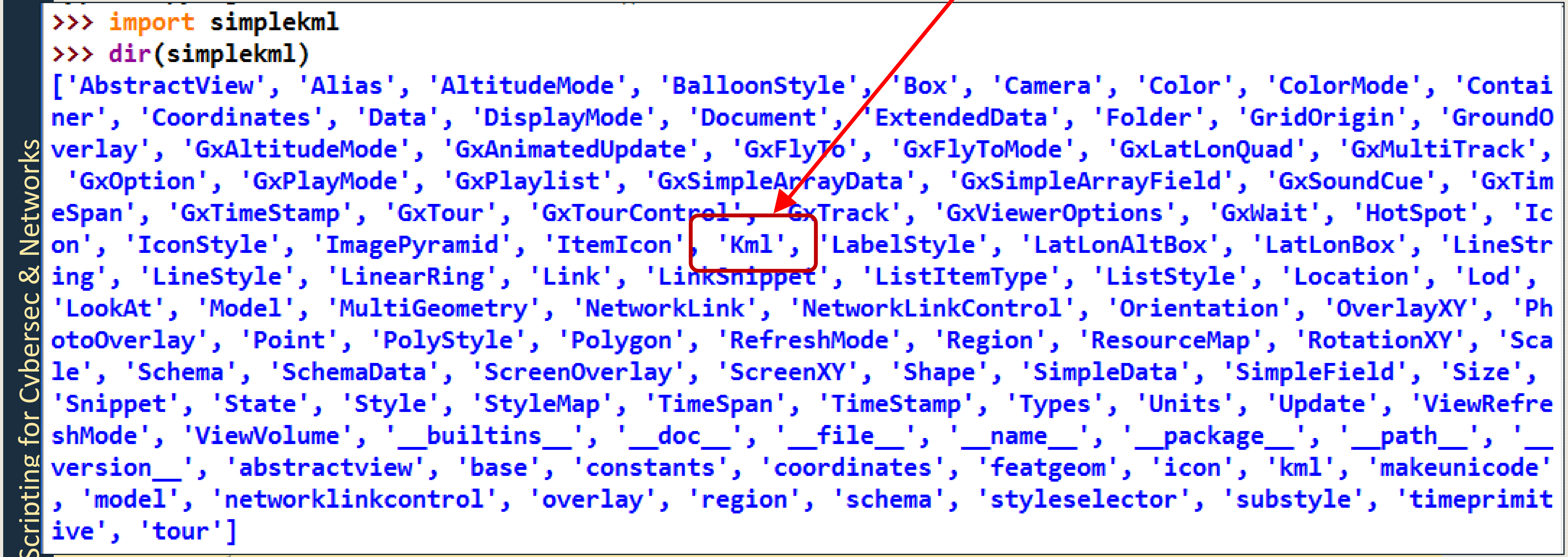
KML root element

Placemark

https://developers.google.com/kml/documentation/kml_tut

Creating KML with Python

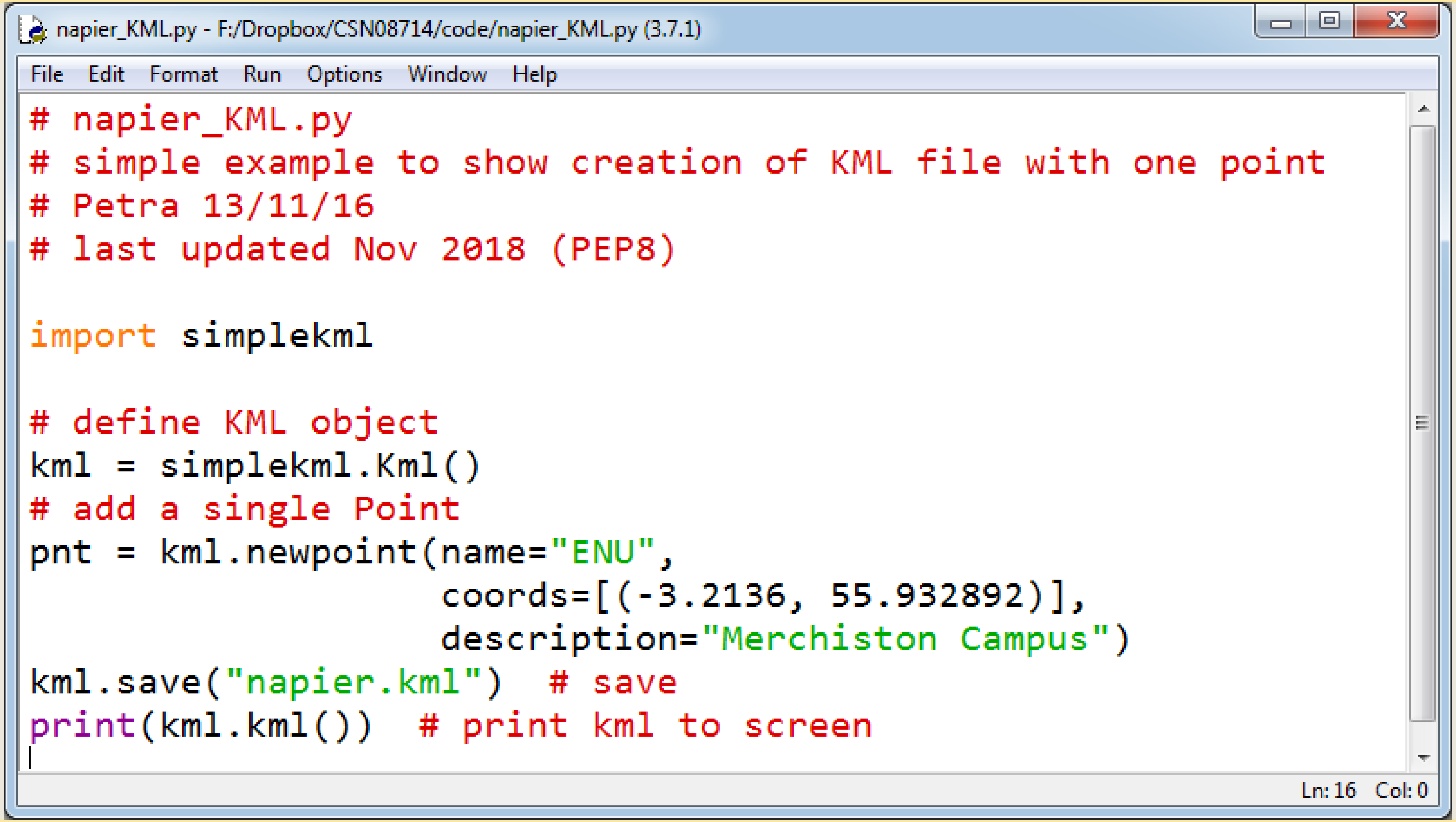
- Use **simplekml** module
- Remember to **pip install** it first



```
>>> import simplekml
>>> dir(simplekml)
['AbstractView', 'Alias', 'AltitudeMode', 'BalloonStyle', 'Box', 'Camera', 'Color', 'ColorMode', 'Contai
ner', 'Coordinates', 'Data', 'DisplayMode', 'Document', 'ExtendedData', 'Folder', 'GridOrigin', 'GroundO
verlay', 'GxAltitudeMode', 'GxAnimatedUpdate', 'GxFlyTo', 'GxFlyToMode', 'GxLatLonQuad', 'GxMultiTrack',
 'GxOption', 'GxPlayMode', 'GxPlaylist', 'GxSimpleArrayData', 'GxSimpleArrayField', 'GxSoundCue', 'GxTim
eSpan', 'GxTimeStamp', 'GxTour', 'GxTourControl', 'GxTrack', 'GxViewerOptions', 'GxWait', 'HotSpot', 'Ic
on', 'IconStyle', 'ImagePyramid', 'ItemIcon', 'Kml', 'LabelStyle', 'LatLonAltBox', 'LatLonBox', 'LineStr
ing', 'LineStyle', 'LinearRing', 'Link', 'LinkSnippet', 'ListItemType', 'ListStyle', 'Location', 'Lod',
'LookAt', 'Model', 'MultiGeometry', 'NetworkLink', 'NetworkLinkControl', 'Orientation', 'OverlayXY', 'Ph
otoOverlay', 'Point', 'PolyStyle', 'Polygon', 'RefreshMode', 'Region', 'ResourceMap', 'RotationXY', 'Sca
le', 'Schema', 'SchemaData', 'ScreenOverlay', 'ScreenXY', 'Shape', 'SimpleData', 'SimpleField', 'Size',
'Snippet', 'State', 'Style', 'StyleMap', 'TimeSpan', 'TimeStamp', 'Types', 'Units', 'Update', 'ViewRefre
shMode', 'ViewVolume', '__builtins__', '__doc__', '__file__', '__name__', '__package__', '__path__', '__
version__', 'abstractview', 'base', 'constants', 'coordinates', 'featgeom', 'icon', 'kml', 'makeunicode'
, 'model', 'networklinkcontrol', 'overlay', 'region', 'schema', 'styleselector', 'substyle', 'timeprimit
ive', 'tour']
```

Kml function to create a kml object

Example: KML with Python

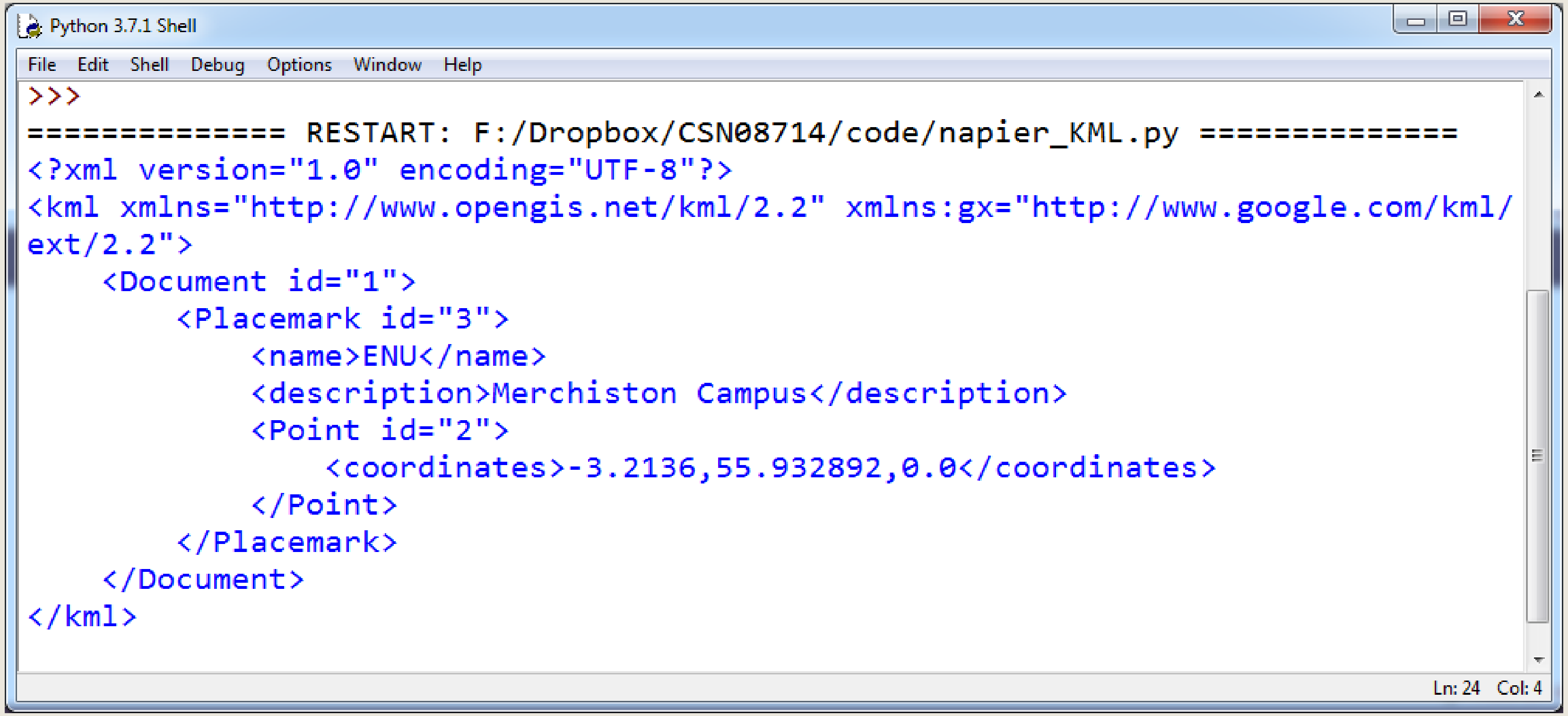


napier_KML.py - F:/Dropbox/CSN08714/code/napier_KML.py (3.7.1)
File Edit Format Run Options Window Help
Ln: 16 Col: 0

```
# napier_KML.py
# simple example to show creation of KML file with one point
# Petra 13/11/16
# last updated Nov 2018 (PEP8)

import simplekml

# define KML object
kml = simplekml.Kml()
# add a single Point
pnt = kml.newpoint(name="ENU",
                   coords=[(-3.2136, 55.932892)],
                   description="Merchiston Campus")
kml.save("napier.kml")  # save
print(kml.kml())  # print kml to screen
```

Example: KML with Python

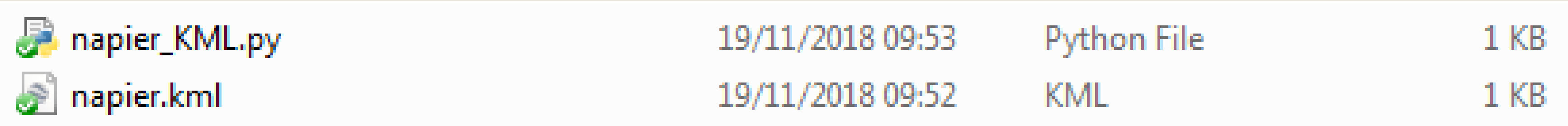
Results of running napier_KML.py:



```
>>>
============== RESTART: F:/Dropbox/CSN08714/code/napier_KML.py ==============
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2" xmlns:gx="http://www.google.com/kml/
ext/2.2">
    <Document id="1">
        <Placemark id="3">
            <name>ENU</name>
            <description>Merchiston Campus</description>
            <Point id="2">
                <coordinates>-3.2136,55.932892,0.0</coordinates>
            </Point>
        </Placemark>
    </Document>
</kml>
```

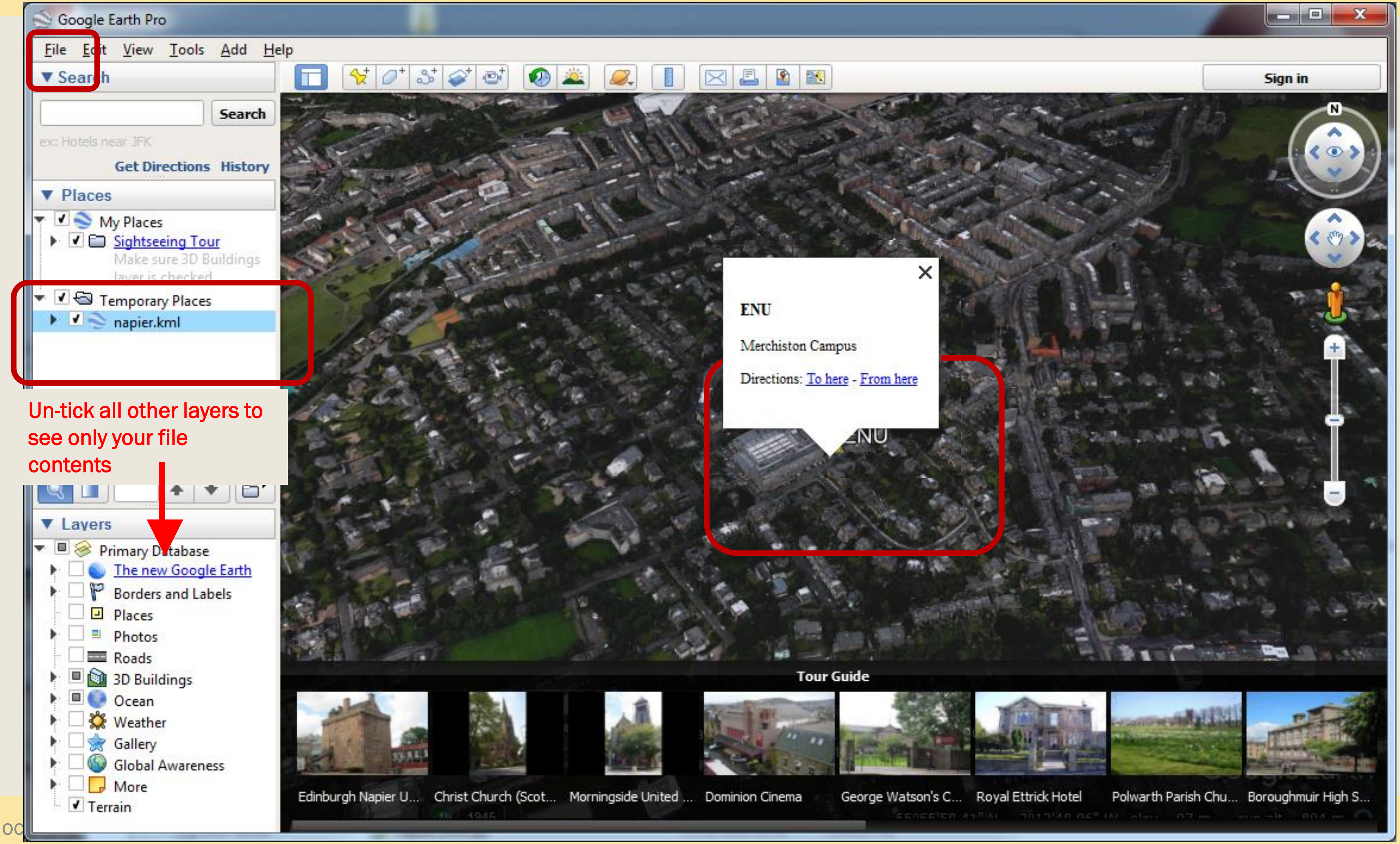
Output to screen

kml File with output created in same directory as script



napier_KML.py	19/11/2018 09:53	Python File	1 KB
napier.kml	19/11/2018 09:52	KML	1 KB

Now open the file in google earth



ENU

Merchiston Campus

Directions: To here - From here

Un-tick all other layers to see only your file contents

Now open the file in google earth

- Open google earth
  - *(install it if necessary - you can install to and run from a portable drive if you wish)*
- File > open, open napier.kml
- Google will "fly" you there, show:
  - *yellow pushpin marker with name*
  - *Description in popup when you click pin*

FAQ

- How to write many points?
  - *Use a "for" loop over the points to be created*
- How to change the icon?
  - *E.g.*

```
pnt.style.iconstyle.scale = 2  # Icon twice as big
pnt.style.iconstyle.icon.href =
'http://maps.google.com/mapfiles/kml/shapes/info-i.png'
                                                          # different icon
```

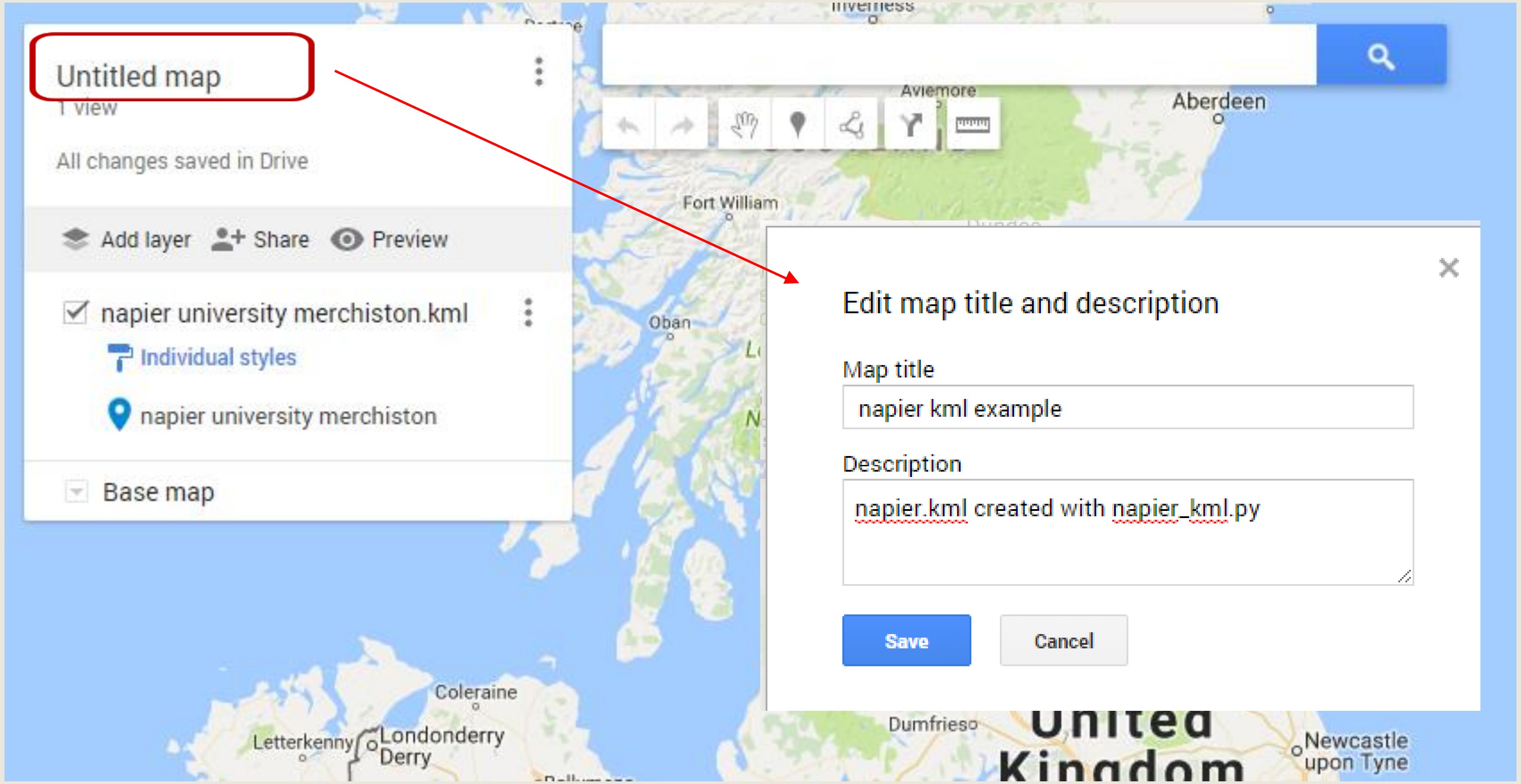
Top Tip:
To find out more about customising the markers, creating tours etc:
simplekml documentation at http://simplekml.readthedocs.io/en/latest/ .
KML tutorial: https://developers.google.com/kml/documentation/kml_tut.

To open a kml in google maps

- You need to use "my maps"
- Go to https://www.google.com/maps/d/ (you may need to log into google)
- Click "create new map"
- Click "Import"

To open a kml in google maps

- After map is open, click on the title "Untitled map" to edit



Edit map title and description

Map title: napier kml example

Description: napier.kml created with napier_kml.py

Alternative: generate static map to auto-open

- Google maps also allows the creation of a static map where all necessary information is given as part of the url
- E.g. https://maps.googleapis.com/maps/api/staticmap?autoscale=2&size=640x640&maptype=roadmap&format=png&visual_refresh=true&markers=label:1%7C55.933064,-3.213589&markers=label:2%7C55.932510,-3.214618&markers=label:3%7C55.928337,-3.837097
- See e.g. https://staticmapmaker.com/google/.

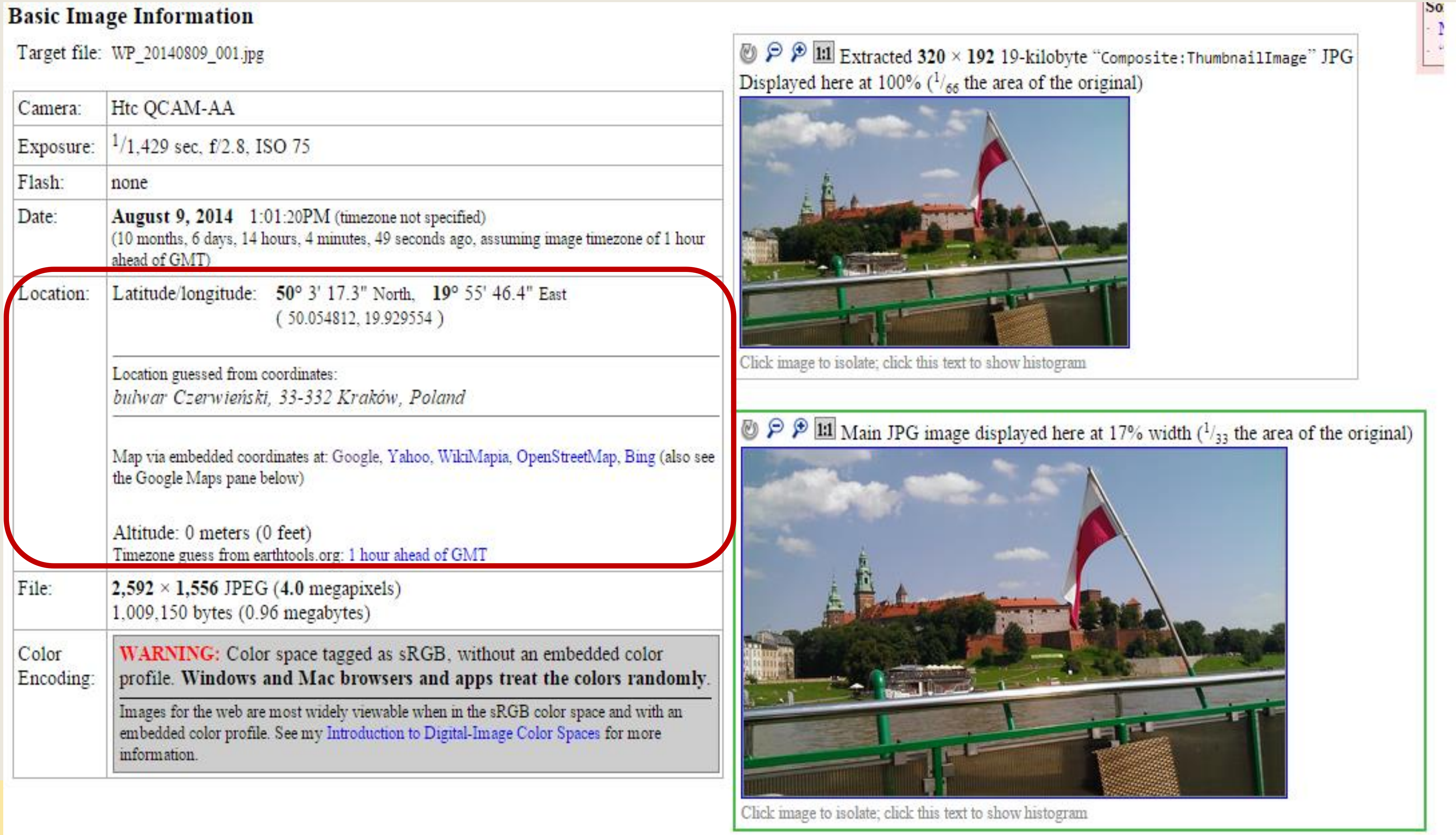
EXIF Information with Python

EXIF photo info

http://regex.info/exif.cgi

Every digital camera stores info about photo within jpg header

"EXIF" contains location if known



Basic Image Information

Target file: WP_20140809_001.jpg

Camera:	Htc QCAM-AA
Exposure:	1/1,429 sec, f/2.8, ISO 75
Flash:	none
Date:	**August 9, 2014** 1:01:20PM (timezone not specified) (10 months, 6 days, 14 hours, 4 minutes, 49 seconds ago, assuming image timezone of 1 hour ahead of GMT)
Location:	Latitude/longitude: **50° 3' 17.3"** North, **19° 55' 46.4"** East (50.054812, 19.929554) Location guessed from coordinates: *bulwar Czerwieński, 33-332 Kraków, Poland* Map via embedded coordinates at: Google, Yahoo, WikiMapia, OpenStreetMap, Bing (also see the Google Maps pane below) Altitude: 0 meters (0 feet) Timezone guess from earthtools.org: 1 hour ahead of GMT
File:	**2,592** × **1,556** JPEG (**4.0** megapixels) 1,009,150 bytes (0.96 megabytes)
Color Encoding:	WARNING: Color space tagged as sRGB, without an embedded color profile. **Windows and Mac browsers and apps treat the colors randomly.** Images for the web are most widely viewable when in the sRGB color space and with an embedded color profile. See my Introduction to Digital-Image Color Spaces for more information.

Extracted **320** × **192** 19-kilobyte "Composite:ThumbnailImage" JPG
Displayed here at 100% (1/66 the area of the original)

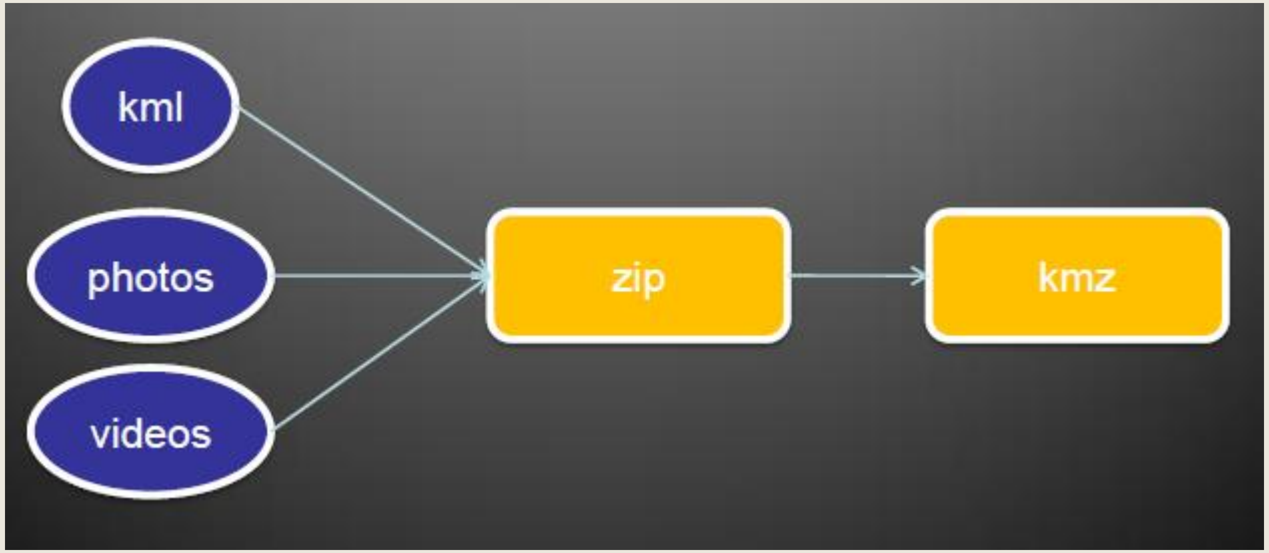
Click image to isolate; click this text to show histogram

Main JPG image displayed here at 17% width (1/33 the area of the original)

Click image to isolate; click this text to show histogram

Mini Project: Photo map

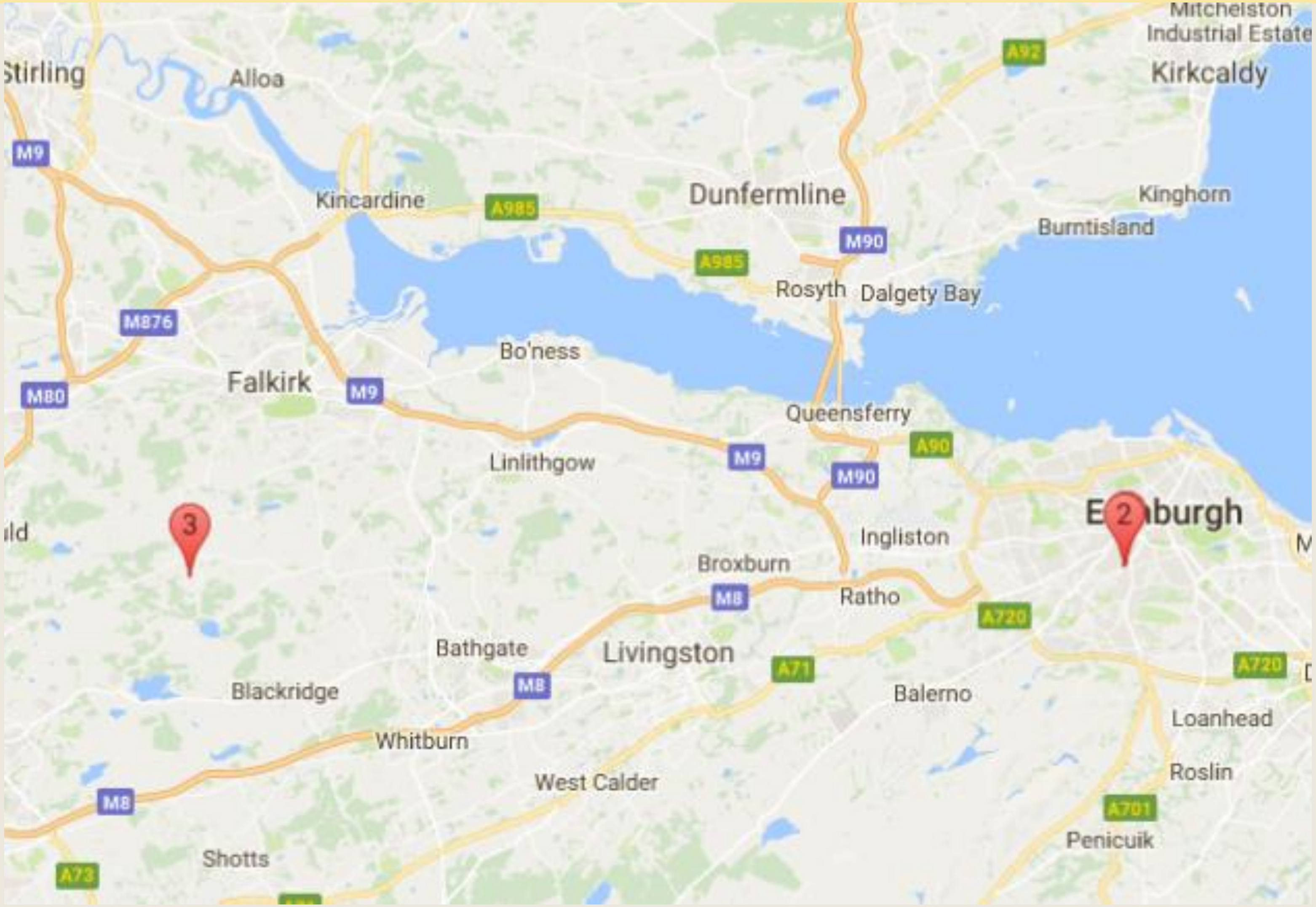
- Folder with photos taken on mobile phone with location on
- Extract location from each photo
  - *exifread module (https://pypi.python.org/pypi/ExifRead)*
  - *PIL, PIL.ExifTags (pip install pillow for these)*
- Create KML / KMZ
- Open in map
  - *It is also possible to import the photos to the map too – learn more about KML/KMZ to find out how.*



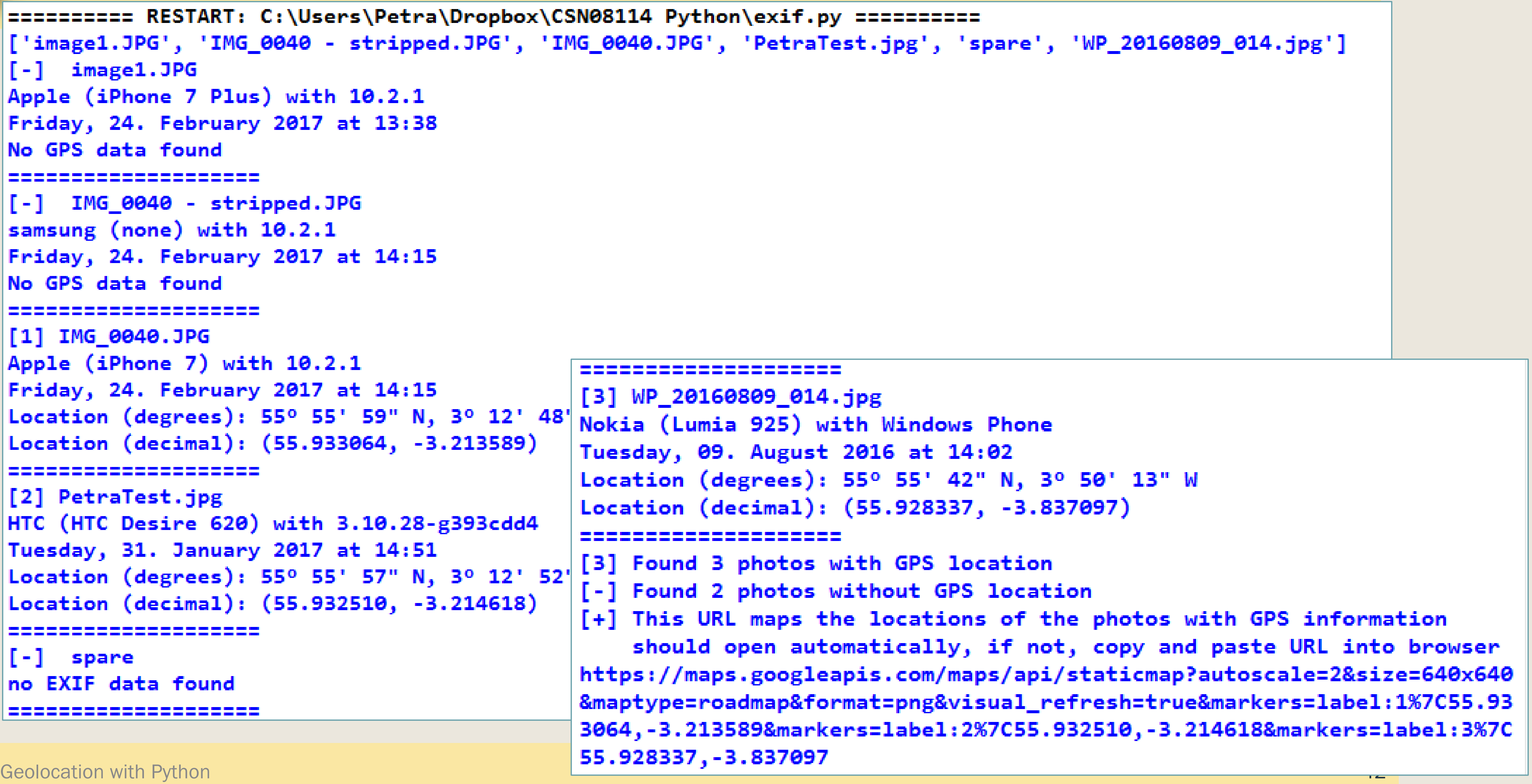
Mini Project: Photo map

- **exif.py** is a working script that extracts EXIF geolocation data from all images in a folder and creates a static map with corresponding markers
- It's very rough and ready and by no means great coding, nor PEP8 compliant, but it works.
- Play around with this in the lab – understand what the various libraries are used for and how the url is generated
- There are many ideas for improving this, for example, by using KMZ instead of a static map you should be able to show thumbnails of the images on the map.

Mini Project: exif.py output example



Mini Project: exif.py output example



```
========== RESTART: C:\Users\Petra\Dropbox\CSN08114 Python\exif.py ==========
['image1.JPG', 'IMG_0040 - stripped.JPG', 'IMG_0040.JPG', 'PetraTest.jpg', 'spare', 'WP_20160809_014.jpg']
[-]  image1.JPG
Apple (iPhone 7 Plus) with 10.2.1
Friday, 24. February 2017 at 13:38
No GPS data found
====================
[-]  IMG_0040 - stripped.JPG
samsung (none) with 10.2.1
Friday, 24. February 2017 at 14:15
No GPS data found
====================
[1] IMG_0040.JPG
Apple (iPhone 7) with 10.2.1
Friday, 24. February 2017 at 14:15
Location (degrees): 55° 55' 59" N, 3° 12' 48'
Location (decimal): (55.933064, -3.213589)
====================
[2] PetraTest.jpg
HTC (HTC Desire 620) with 3.10.28-g393cdd4
Tuesday, 31. January 2017 at 14:51
Location (degrees): 55° 55' 57" N, 3° 12' 52'
Location (decimal): (55.932510, -3.214618)
====================
[-]  spare
no EXIF data found
====================
```

```
====================
[3] WP_20160809_014.jpg
Nokia (Lumia 925) with Windows Phone
Tuesday, 09. August 2016 at 14:02
Location (degrees): 55° 55' 42" N, 3° 50' 13" W
Location (decimal): (55.928337, -3.837097)
====================
[3] Found 3 photos with GPS location
[-] Found 2 photos without GPS location
[+] This URL maps the locations of the photos with GPS information
    should open automatically, if not, copy and paste URL into browser
https://maps.googleapis.com/maps/api/staticmap?autoscale=2&size=640x640
&maptype=roadmap&format=png&visual_refresh=true&markers=label:1%7C55.93
3064,-3.213589&markers=label:2%7C55.932510,-3.214618&markers=label:3%7C
55.928337,-3.837097
```

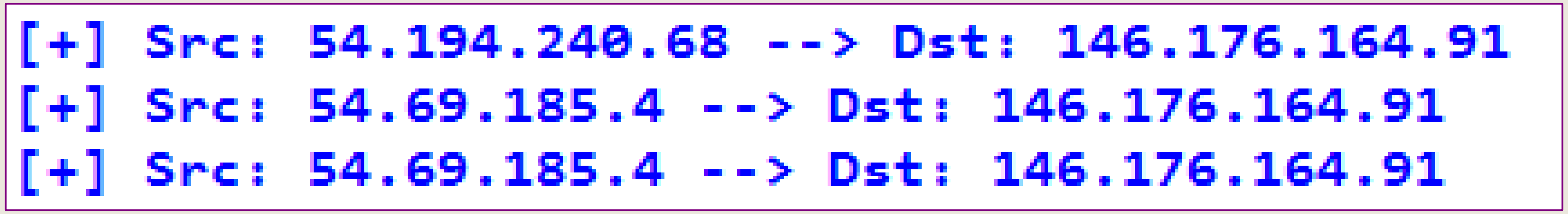
Practical

- Lab 11
- Look up geolocation info for IP addresses
- KML and Photo map example
- Map of IP addresses

Remember the networking lab

pcap_downloads.py script was given

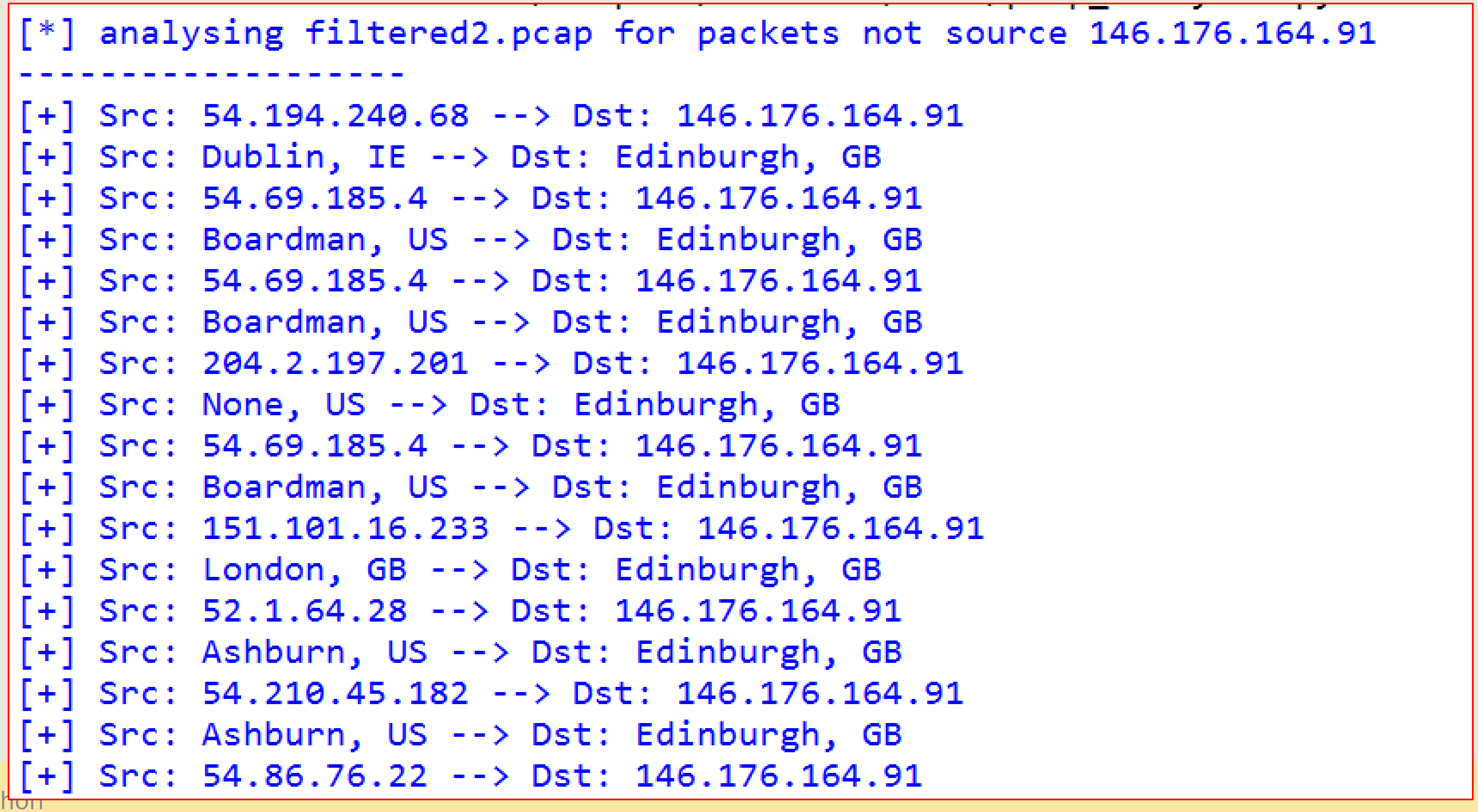
One Exercise was to write a new script **pcap_exclude.py** that will show traffic source and destination for each packet from a pcap wireshark network capture in the format:



```
[+] Src: 54.194.240.68 --> Dst: 146.176.164.91
[+] Src: 54.69.185.4 --> Dst: 146.176.164.91
[+] Src: 54.69.185.4 --> Dst: 146.176.164.91
```

Now add geolocation info

Expand pcap_exclude.py to use pygeoip to generate output in the format:

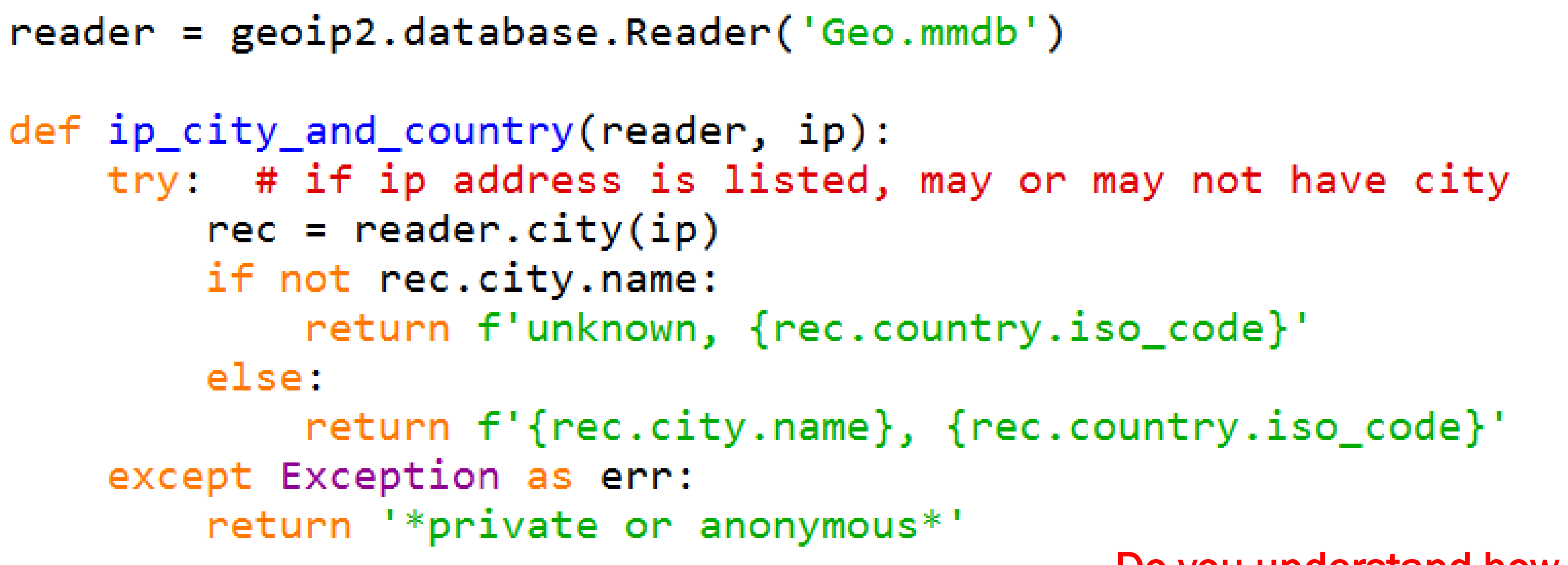


```
[*] analysing filtered2.pcap for packets not source 146.176.164.91
-------------------
[+] Src: 54.194.240.68 --> Dst: 146.176.164.91
[+] Src: Dublin, IE --> Dst: Edinburgh, GB
[+] Src: 54.69.185.4 --> Dst: 146.176.164.91
[+] Src: Boardman, US --> Dst: Edinburgh, GB
[+] Src: 54.69.185.4 --> Dst: 146.176.164.91
[+] Src: Boardman, US --> Dst: Edinburgh, GB
[+] Src: 204.2.197.201 --> Dst: 146.176.164.91
[+] Src: None, US --> Dst: Edinburgh, GB
[+] Src: 54.69.185.4 --> Dst: 146.176.164.91
[+] Src: Boardman, US --> Dst: Edinburgh, GB
[+] Src: 151.101.16.233 --> Dst: 146.176.164.91
[+] Src: London, GB --> Dst: Edinburgh, GB
[+] Src: 52.1.64.28 --> Dst: 146.176.164.91
[+] Src: Ashburn, US --> Dst: Edinburgh, GB
[+] Src: 54.210.45.182 --> Dst: 146.176.164.91
[+] Src: Ashburn, US --> Dst: Edinburgh, GB
[+] Src: 54.86.76.22 --> Dst: 146.176.164.91
```

How?

How? (part 1)

- Add function to look up city and country code for a given IP address

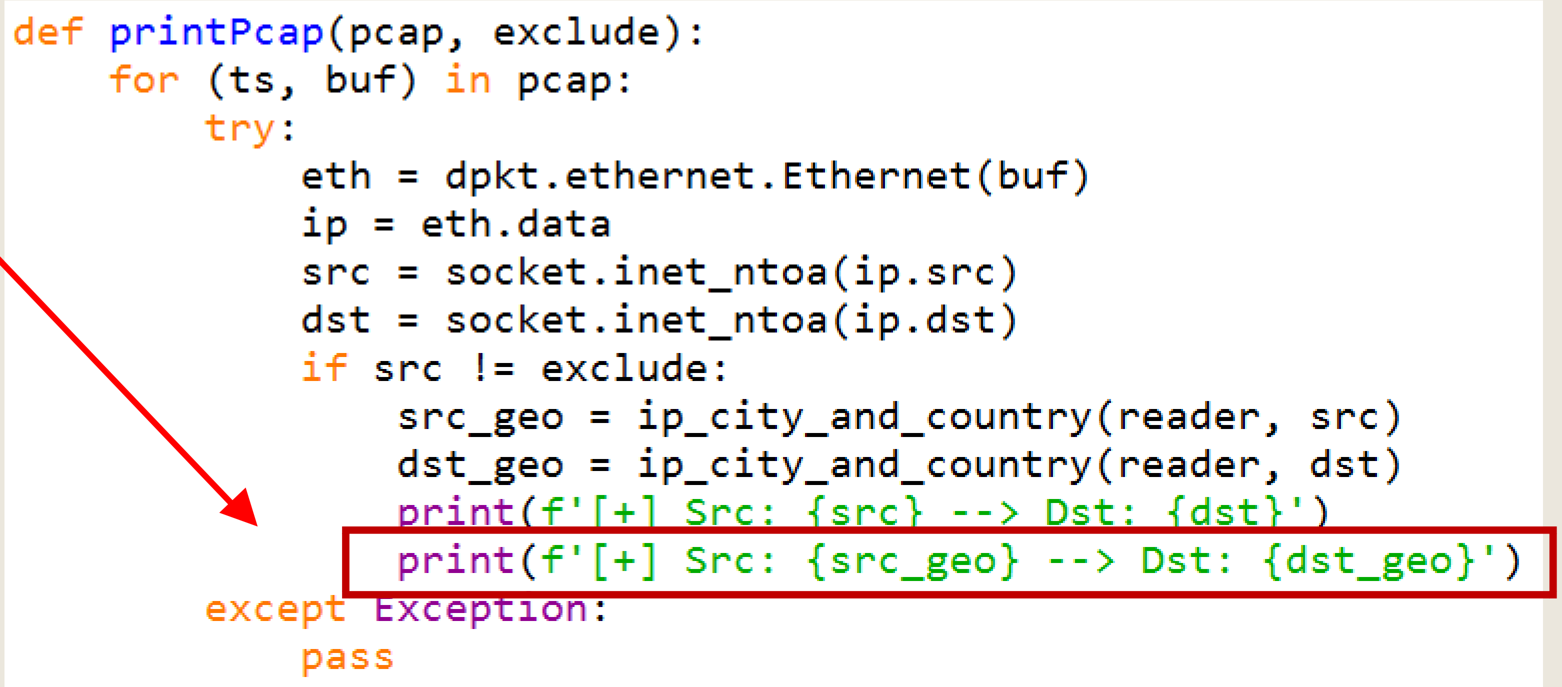


```
reader = geoip2.database.Reader('Geo.mmdb')

def ip_city_and_country(reader, ip):
    try:  # if ip address is listed, may or may not have city
        rec = reader.city(ip)
        if not rec.city.name:
            return f'unknown, {rec.country.iso_code}'
        else:
            return f'{rec.city.name}, {rec.country.iso_code}'
    except Exception as err:
        return '*private or anonymous*'
```

Do you understand how this works?
Try calling the function e.g. with
'146.176.164.91'
'147.176.164.91'
'77.72.112.213'
'127.0.0.1'

How? (part 2)

1. Add a function to the script to look up city and country code for a given IP address (mine was called **ip_city_and_country**)
2. Add a second print statement to existing print function to print the results



```python
def printPcap(pcap, exclude):
    for (ts, buf) in pcap:
        try:
            eth = dpkt.ethernet.Ethernet(buf)
            ip = eth.data
            src = socket.inet_ntoa(ip.src)
            dst = socket.inet_ntoa(ip.dst)
            if src != exclude:
                src_geo = ip_city_and_country(reader, src)
                dst_geo = ip_city_and_country(reader, dst)
                print(f'[+] Src: {src} --> Dst: {dst}')
                print(f'[+] Src: {src_geo} --> Dst: {dst_geo}')
        except Exception:
            pass
```

Putting it all together (1)

- Create your expanded script using the previous slides for guidance
(let's call it pcap_analysis.py)
- Test with filtered2.pcap and filtered3.pcap
- Understand fully how it works
- After part B, extend this by mapping the IP addresses on a map

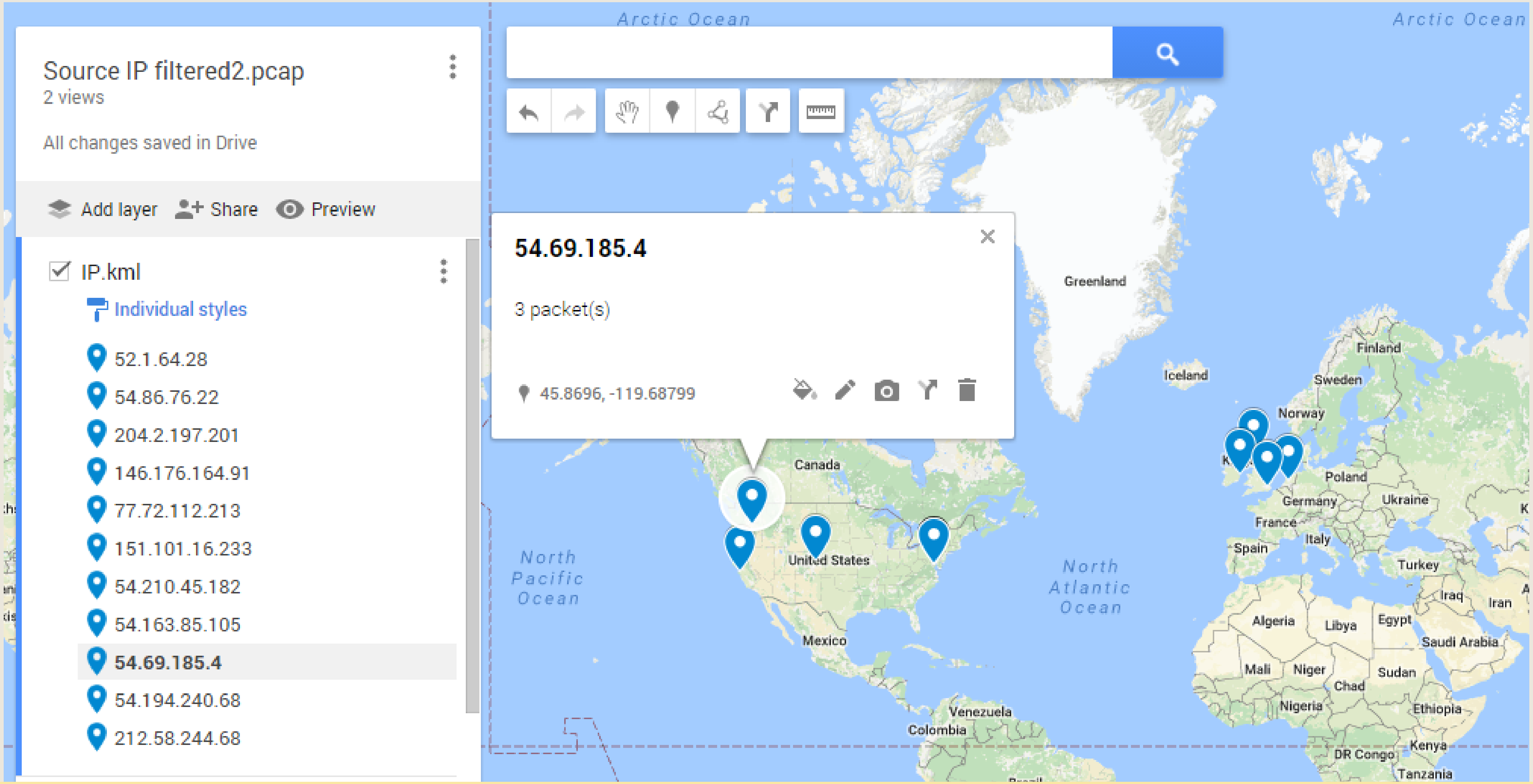
Pcap geolocation analysis - mapping the IP addresses

Create a new script, **pcap_kml.py**, which

- is based on pcap_analysis.py
- Reads a pcap file
- Extracts the IP addresses of the source of each packet
- Makes a dictionary of these addresses, where the IP address is the key and the number of packets the value
  - *Hint: you can use collections.counter() for this*
- Creates a KML file that contains a Placemark for each **source** IP address that has a known geolocation
  - *where the Description is the packet count*
  - *optionally, the description could also include the city and country*
  - *optionally, the icon size (or colour) could be changed to reflect the packet count*

…and the result?

Better in maps than earth (why?)



Source IP filtered2.pcap
2 views

All changes saved in Drive

Add layer Share Preview

☑ IP.kml

Individual styles

- 52.1.64.28
- 54.86.76.22
- 204.2.197.201
- 146.176.164.91
- 77.72.112.213
- 151.101.16.233
- 54.210.45.182
- 54.163.85.105
- **54.69.185.4**
- 54.194.240.68
- 212.58.244.68

54.69.185.4

3 packet(s)

45.8696, -119.68799